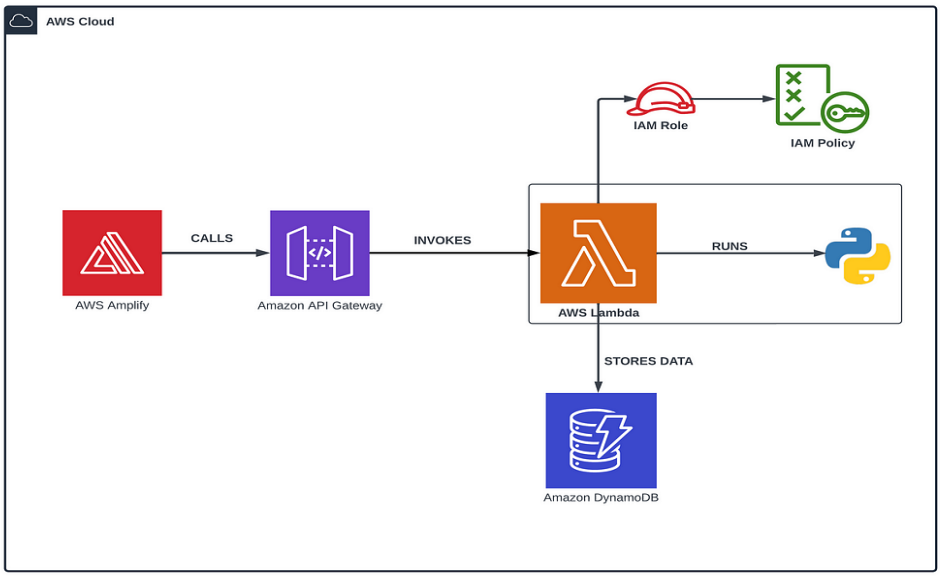
Serverless Web App Development



Project Overview:

This project is a web application that calculates the area of a rectangle. Users can input the length and width of a rectangle, and the application will display the calculated area. The application is built using AWS services such as AWS Amplify, API Gateway, Lambda, and DynamoDB.

Prerequisites:

AWS account

Frontend web application (Index.html file)

Setup Instructions:

- Setting up Dynamo DB
- Setting up Lambda
- Setting up API gateway
- Setting up Frontend web application
- Setting up AWS amplify

Sign into your AWS console

AWS (Amazon Web Services) is a comprehensive cloud computing platform offering a wide range of services for computing, storage, and networking.

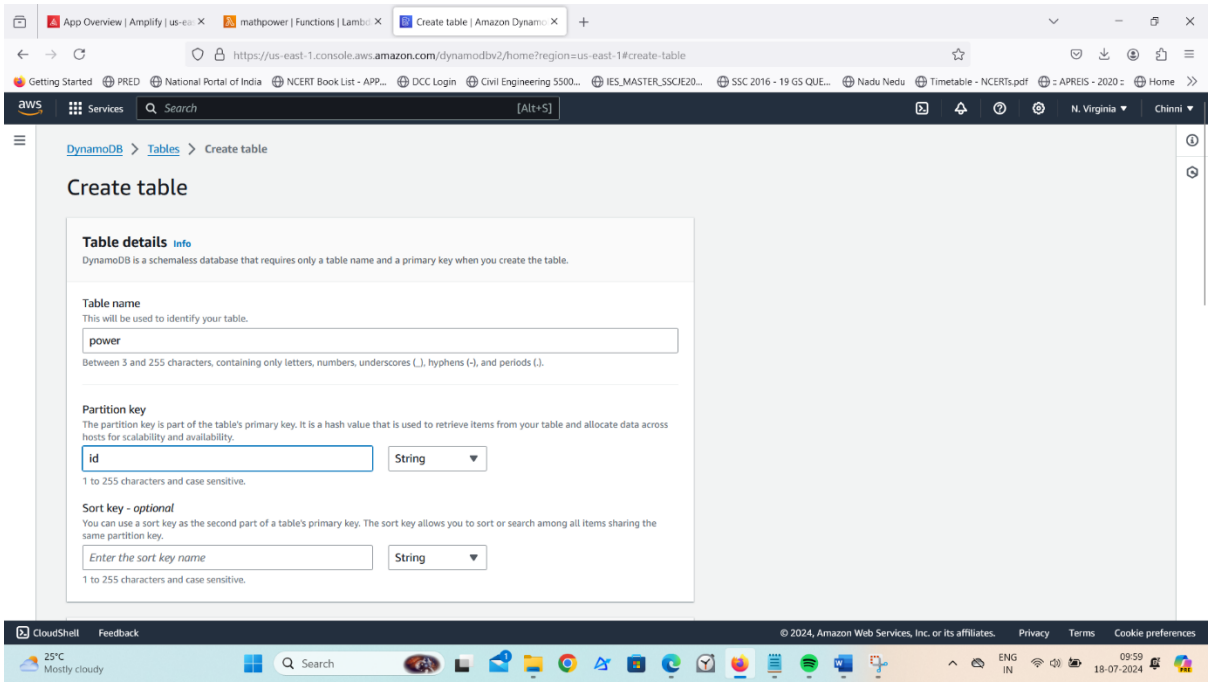
Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.

Step1: Setting up Dynamo DB

On the services search box, search for ‘DynamoDB’ and select the DynamoDB service.

Click on create table.

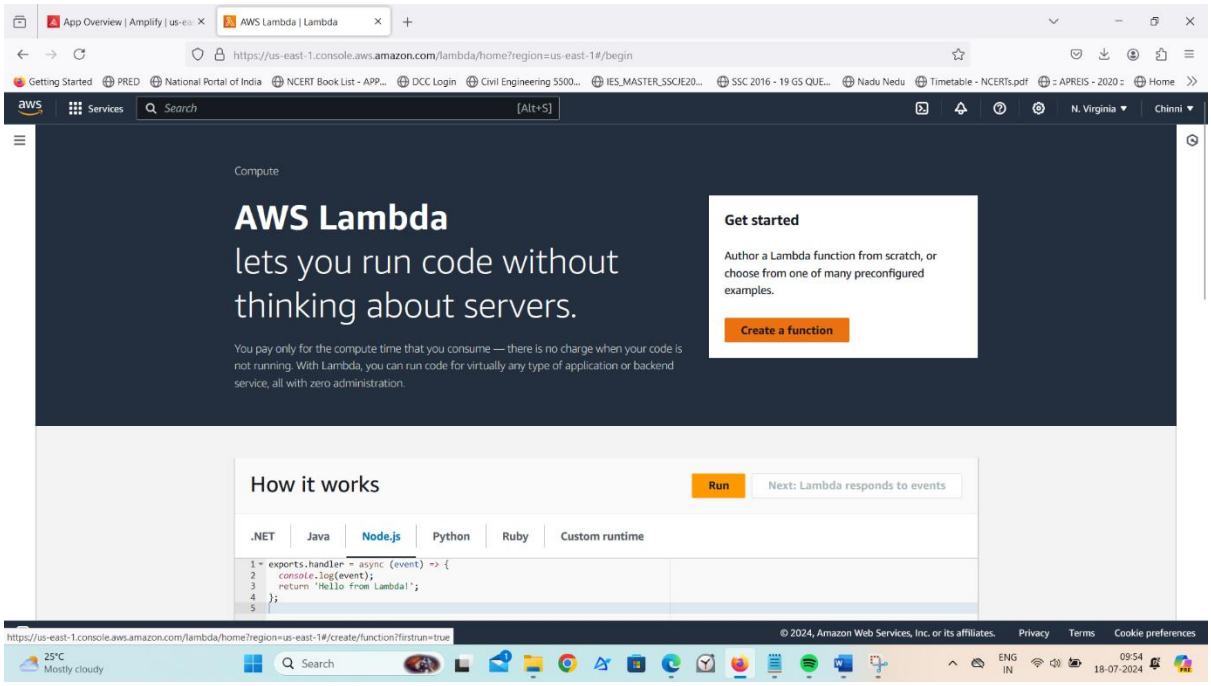
Give the table a name, for ‘Partition key’ input ‘ID’. Leave the rest as default, scroll to the bottom and click on ‘Create table’



AWS Lambda is a serverless compute service that automatically runs your code in response to events and scales the underlying compute resources as needed, without the need to provision or manage servers.

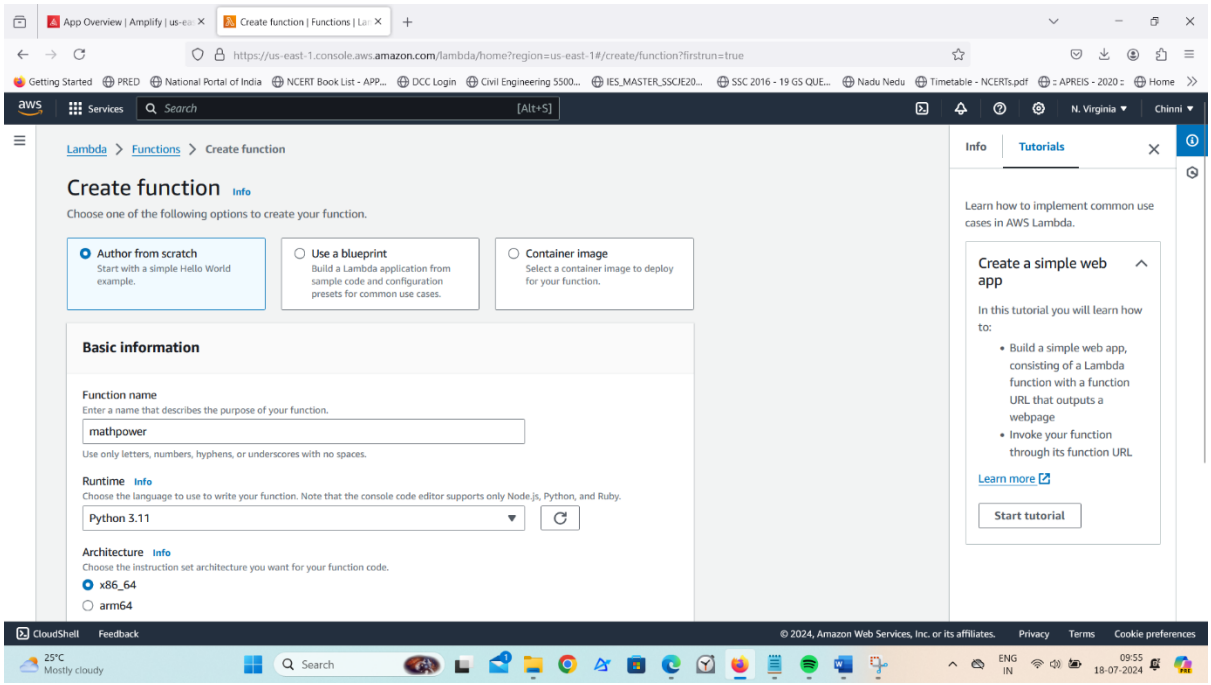
Step2: Setting up Lambda

On the AWS console search bar, type ‘Lambda’ and select the Lambda service.



Click on ‘Create function’.

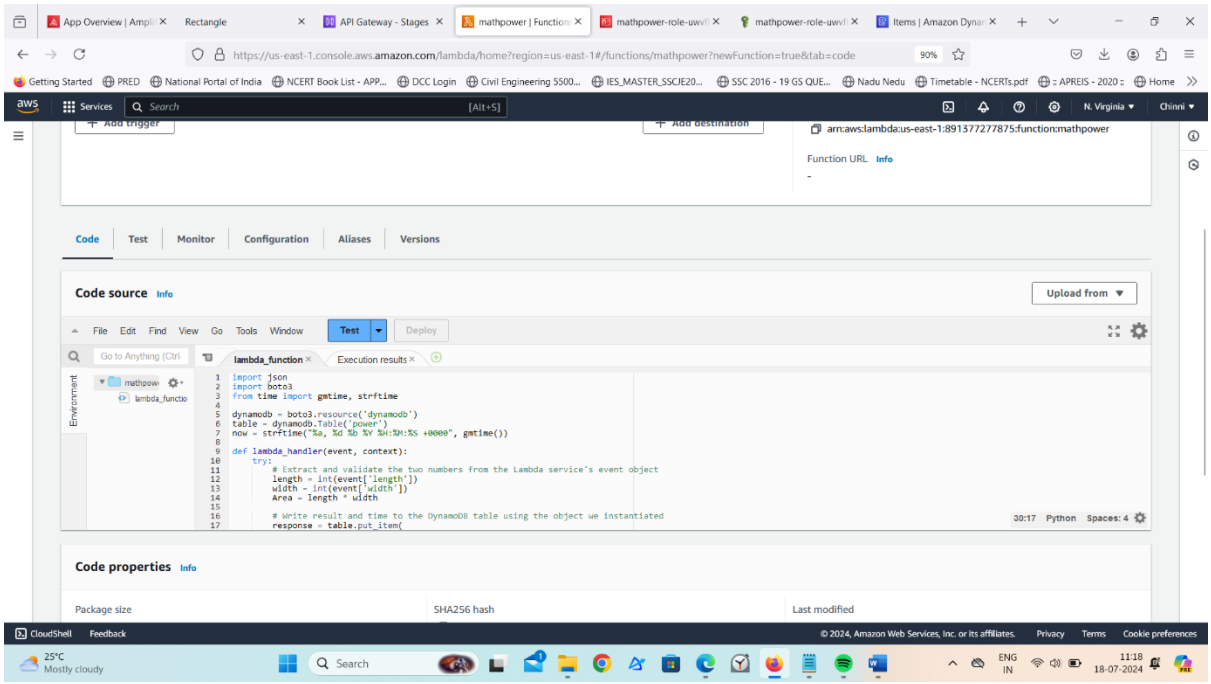
Give the Function name, The Runtime (Latest Python), then scroll down and click on ‘Create function’



Copy the following Lambda function onto your lambda_function.py file. Please give the DynamoDB name (as created above).

```
import json
import boto3
from time import gmtime, strftime

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('power')
now = strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())

def lambda_handler(event, context):
    try:
        length = int(event['length'])
        width = int(event['width'])
        Area = length * width

        response = table.put_item(
            Item={
                'ID': str(Area),
                'LatestGreetingTime': now,
            }
        )
        return {
            'statusCode': 200,
            'body': json.dumps('Your result is ' + str(Area))
```

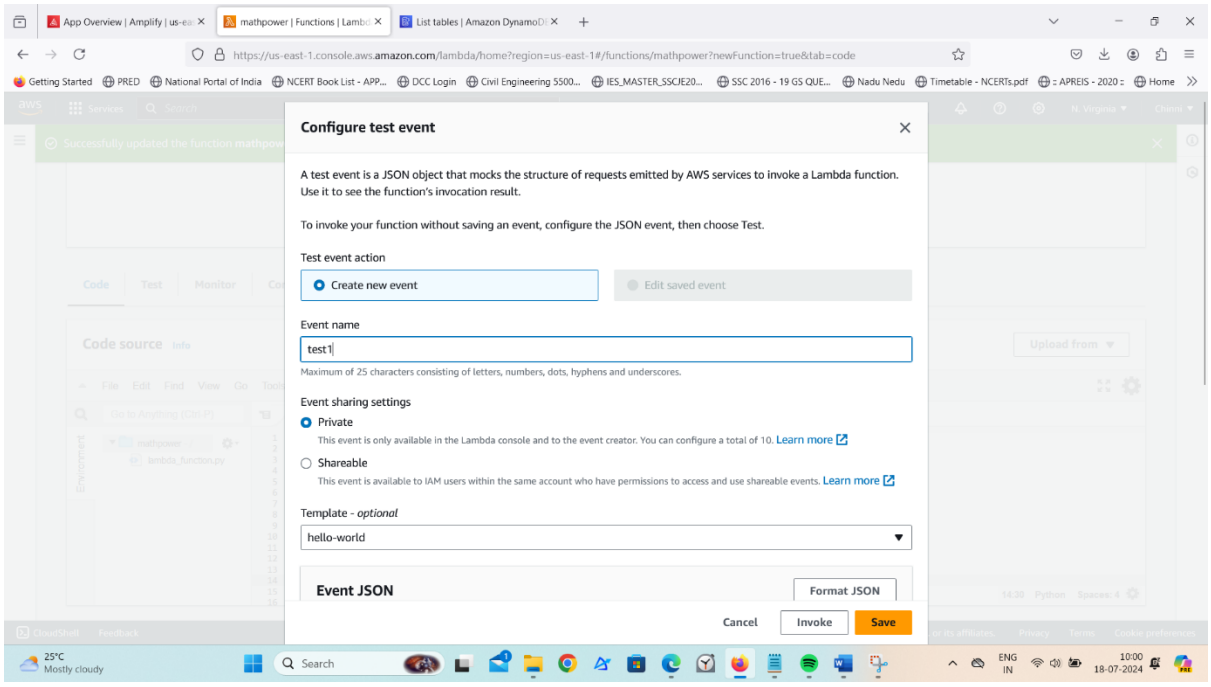
```
    }


except ValueError:
    return {
        'statusCode': 400,
        'body': json.dumps('Error: length and width must be integers.')
    }
except Exception as e:
    return {
        'statusCode': 500,
        'body': json.dumps('Error: ' + str(e))
    }
```

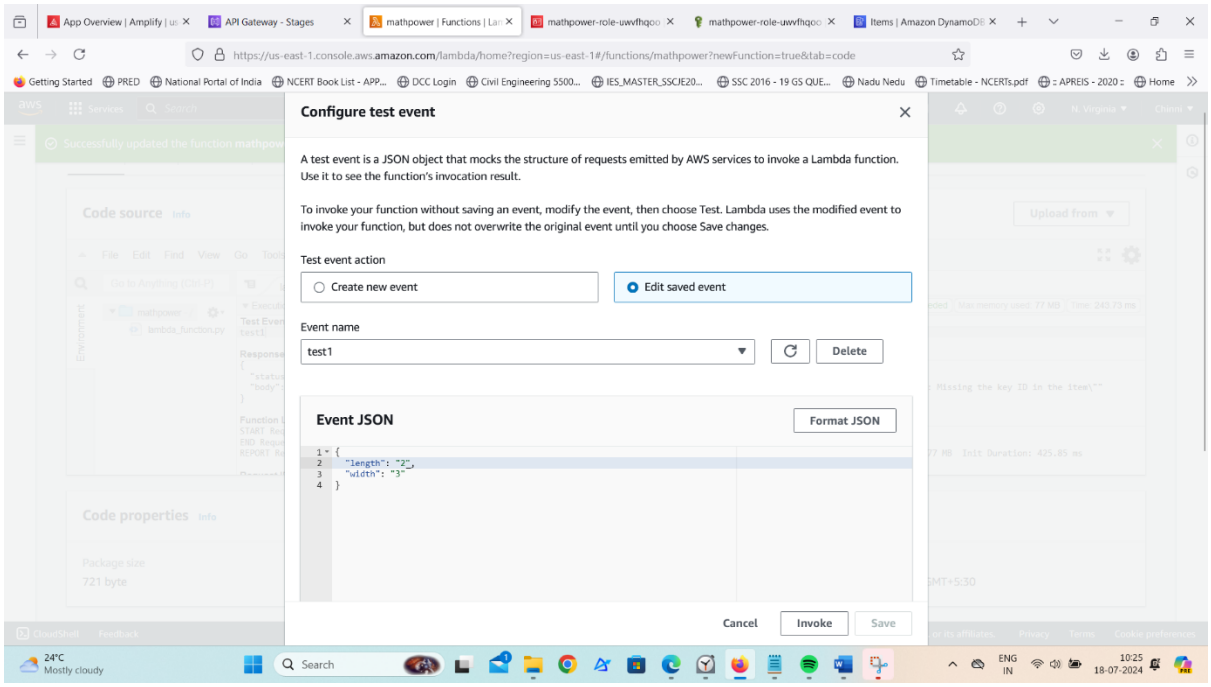


Click on deploy. Then click on test. We will get a window open then give event name and in event give the values of length and width.

Click on save and click on test.

We get our result (2*3=6)

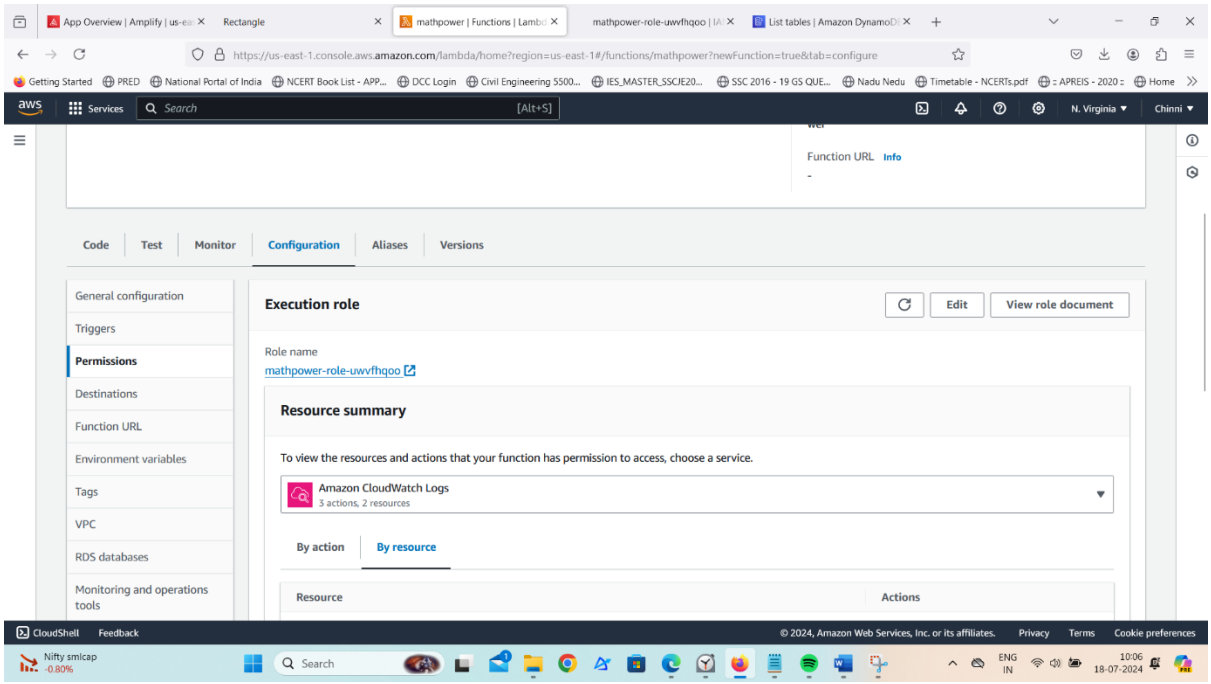


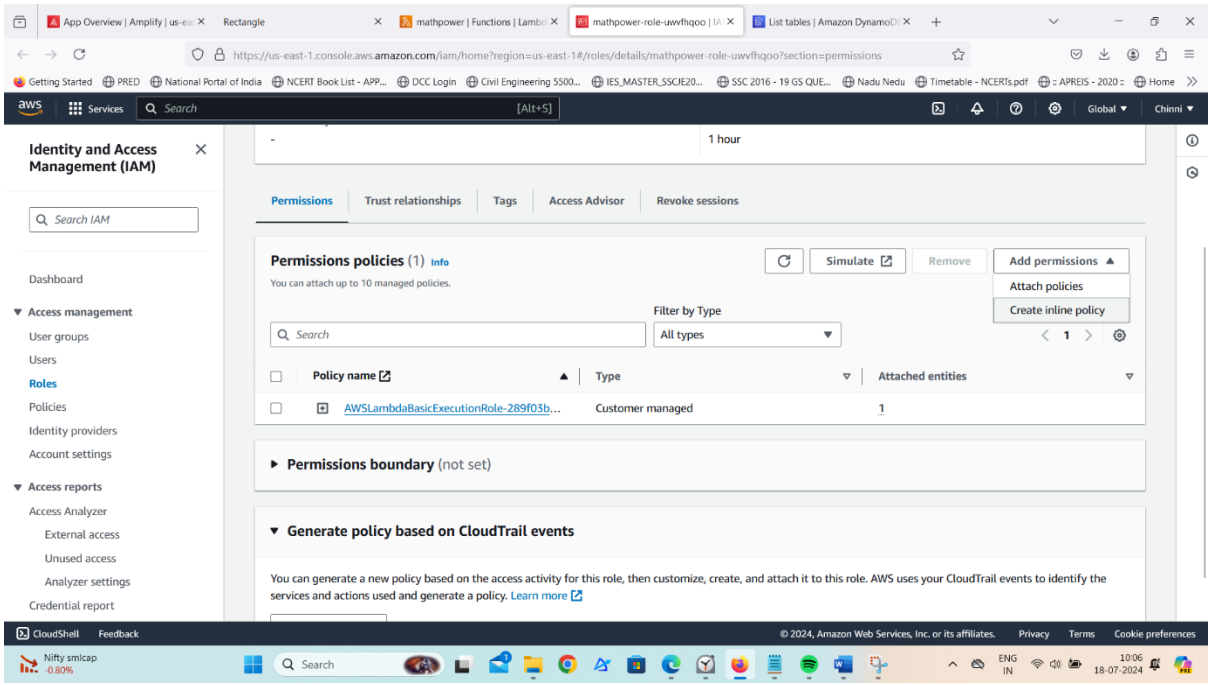


At first test we get an error like access denied for the click on lambda configurations tab in that click on permissions tab.

An IAM role in AWS (Identity and Access Management) defines a set of permissions that grant access to AWS resources. It is used to delegate access to users, applications, or services, ensuring secure and controlled interactions with AWS services and resources.

In permissions tab we can see the IAM role click on that.

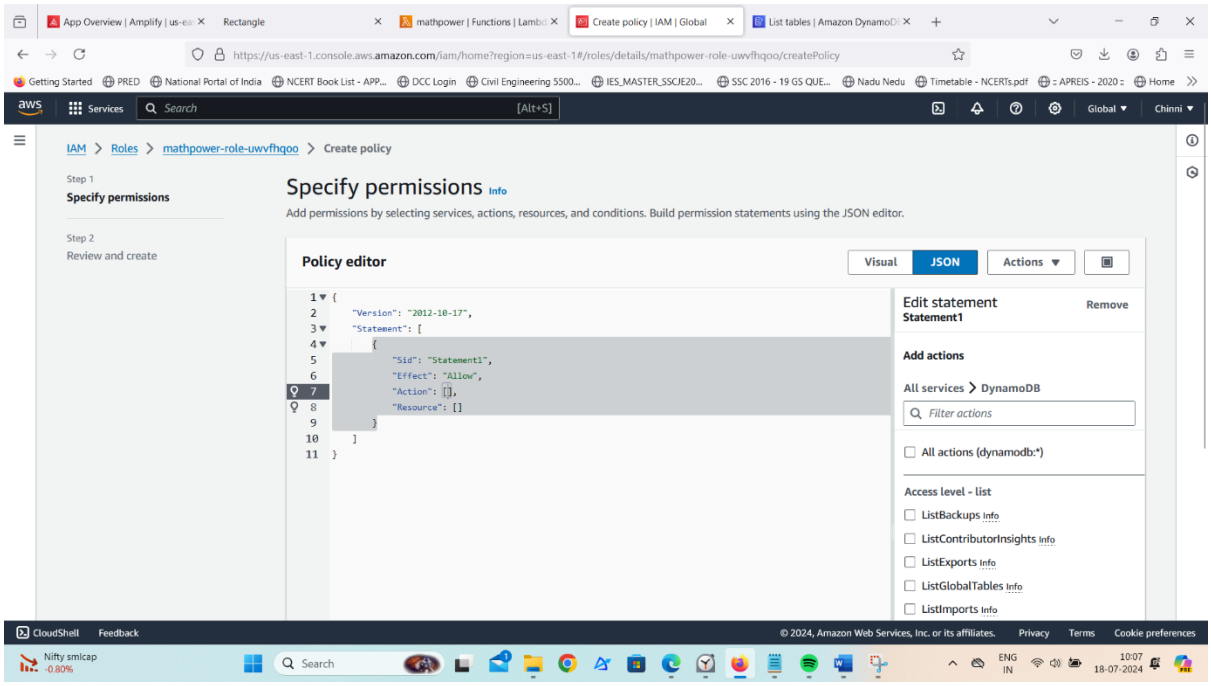




It will open like this, now click on add permissions in that we see create inline policy. Click on it.

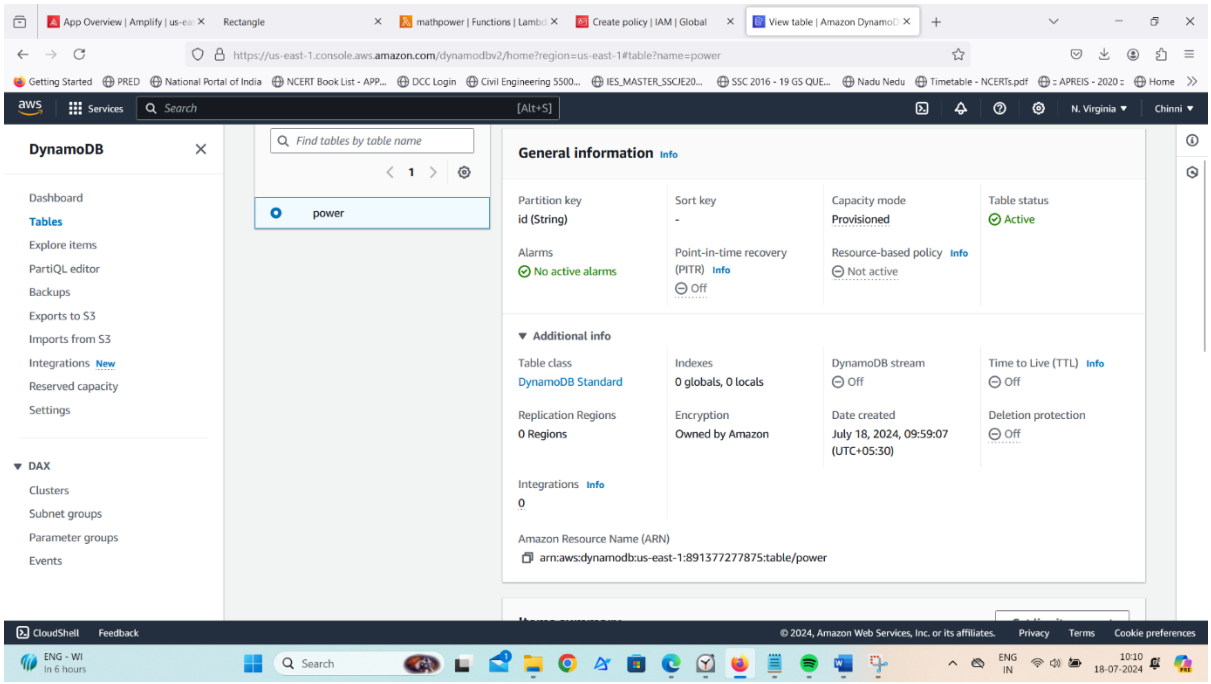
Click on JSON in that search for dynamo DB in all services tab.

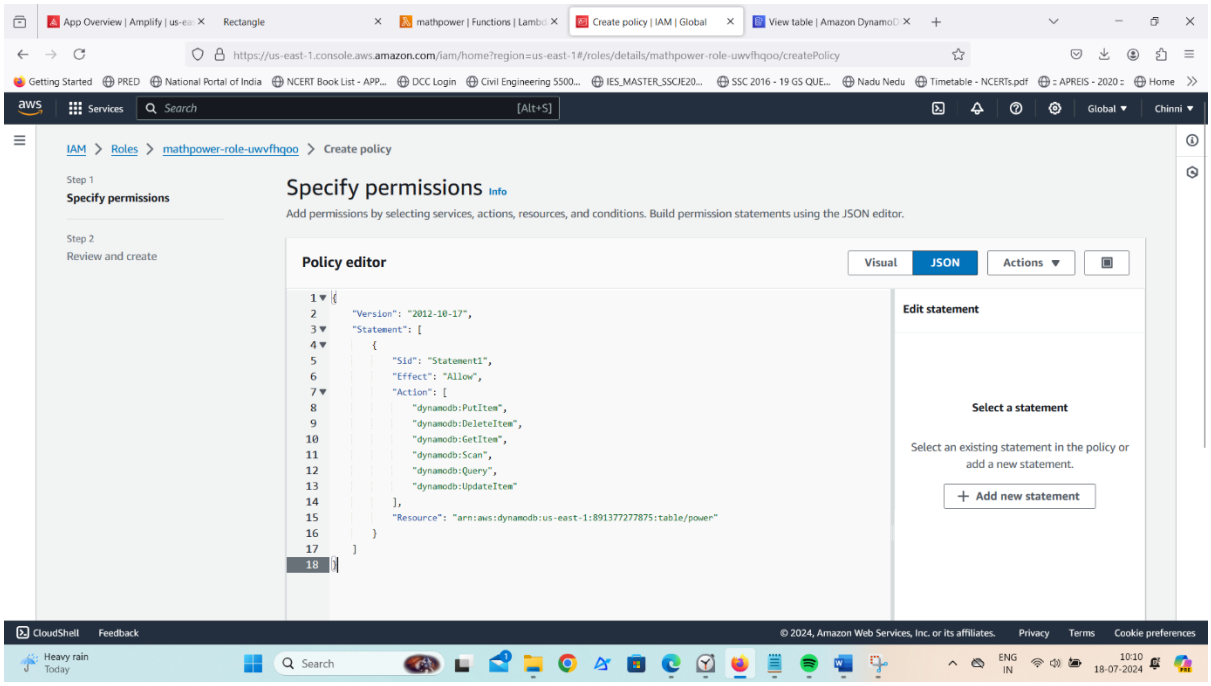
Give policies like put item, delete item, update item, get item, scan, query.



At resources copy our dynamo DB table arn (amazon resource name).

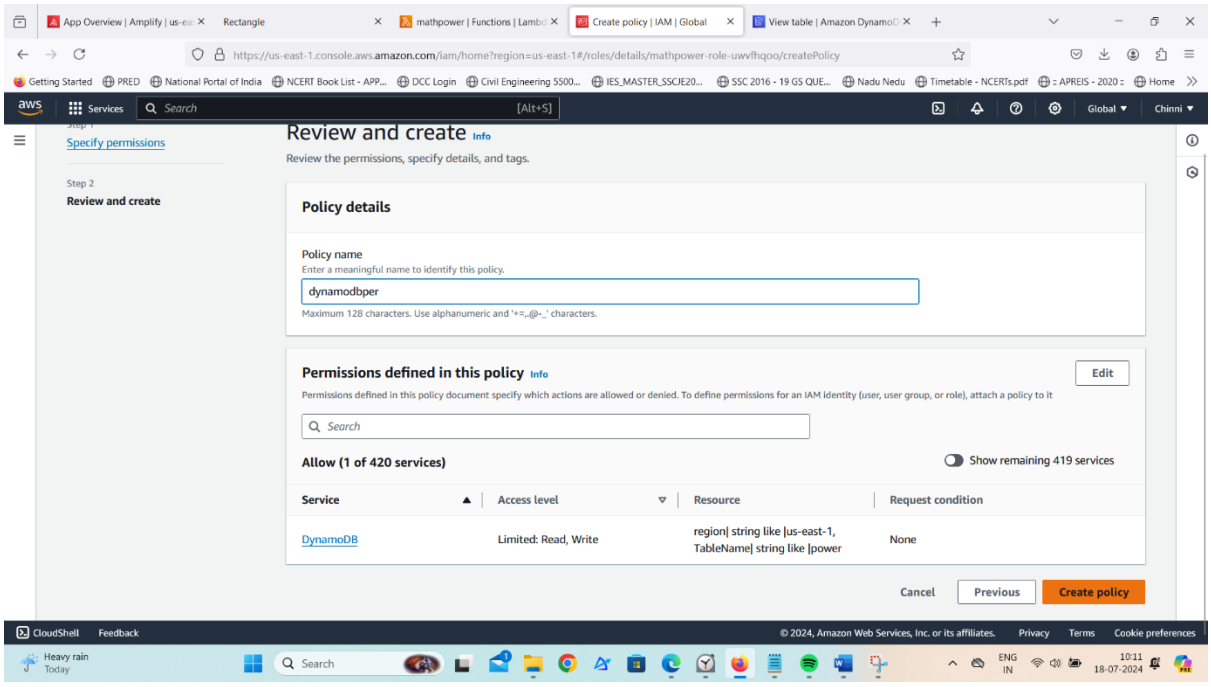
Click on our table we get general information under that we have additional info drag that tab we see our table arn paste that on policy.

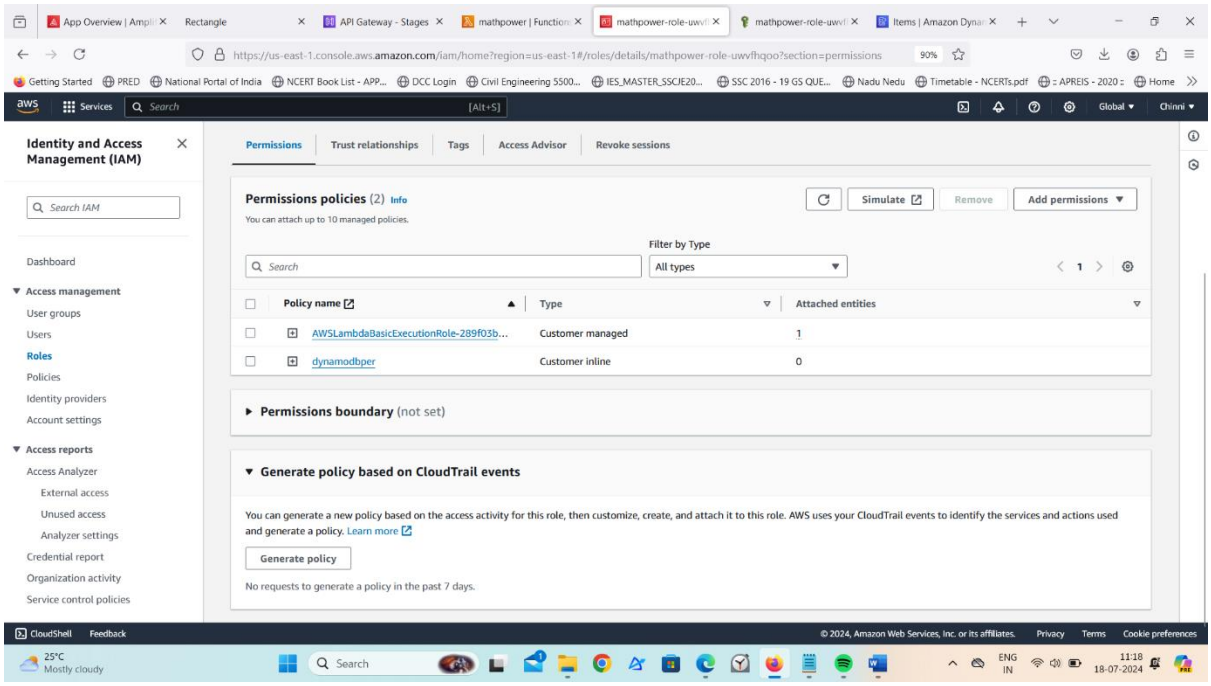




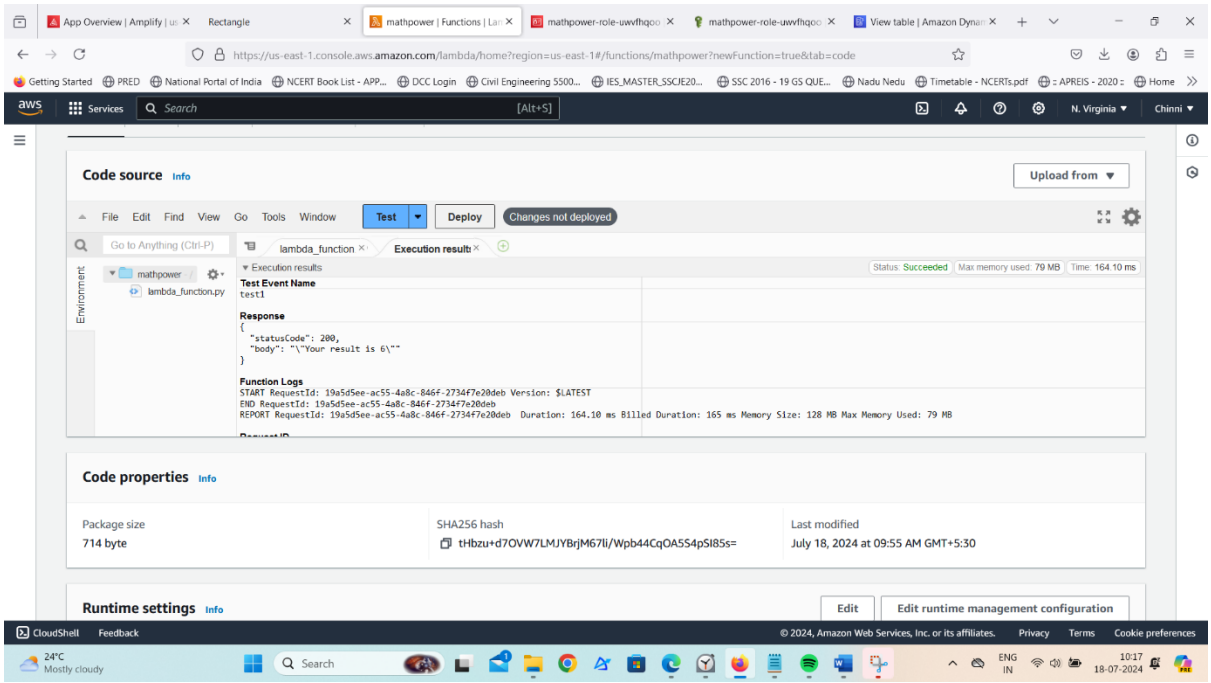
Click on next.

Give policy name and click on create policy.

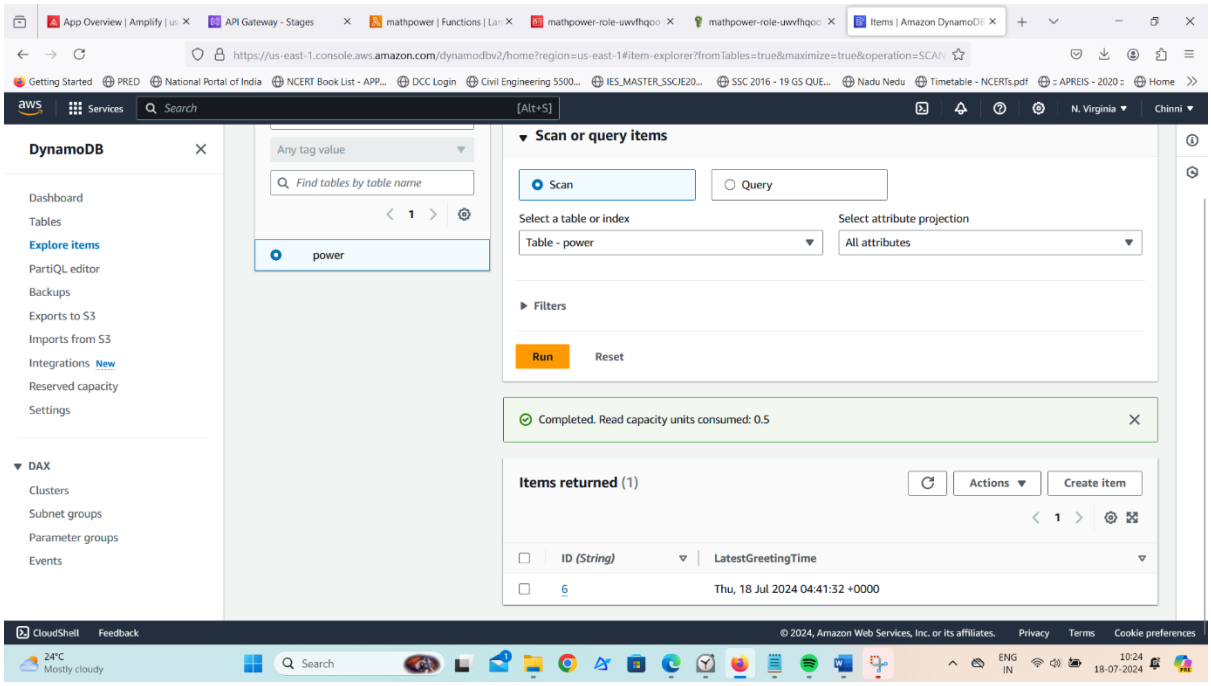




Now go to lambda click on test we get result.



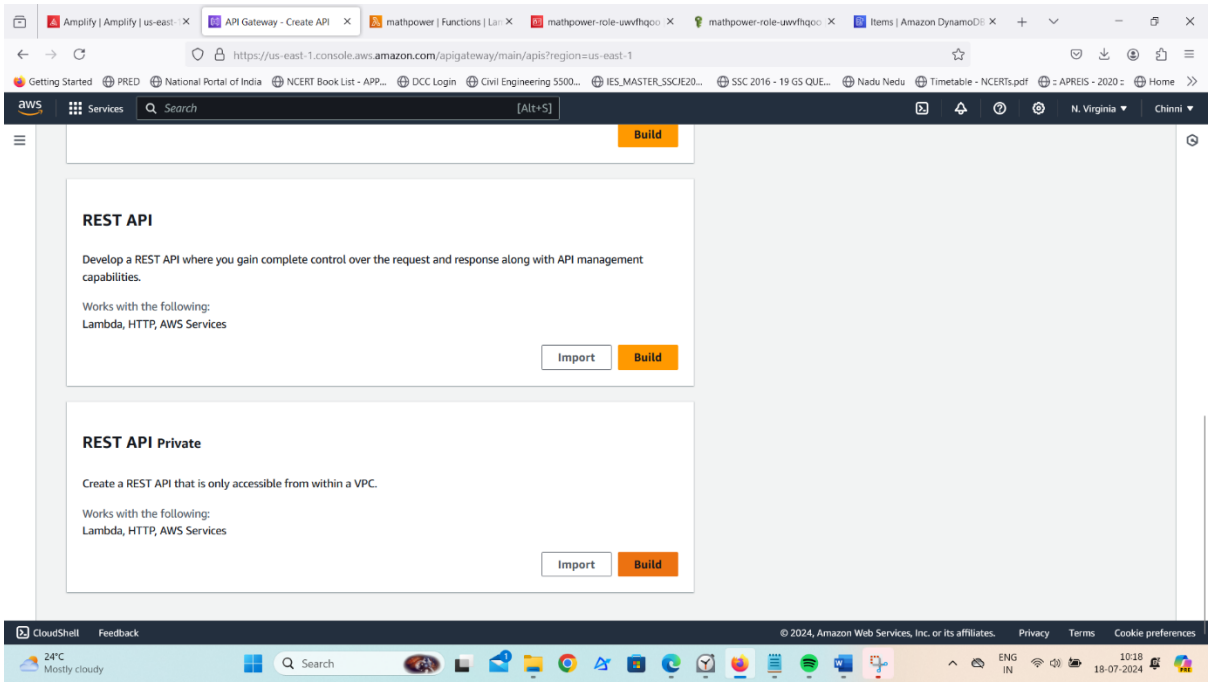
we can see our data is saved on our created dynamo DB table.



Step 3: Setting up API gateway

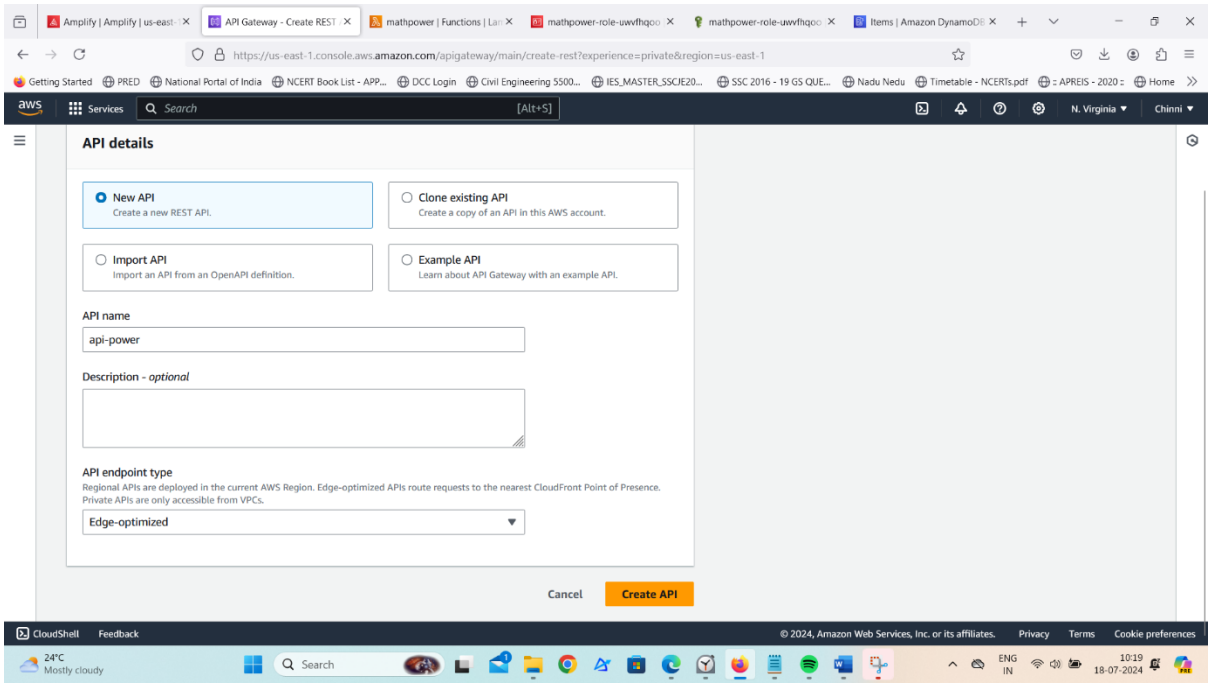
API Gateway in AWS is a fully managed service that enables developers to create, publish, maintain, monitor, and secure APIs at any scale. It acts as a front door for applications to access data, business logic, or functionality from backend services, including Lambda functions, AWS services, or other HTTP endpoints.

- On the AWS services search box, enter ‘API’ and select ‘API Gateway’ that appears
- In the list for ‘Choose an API type’, select ‘Build’ for ‘REST API’

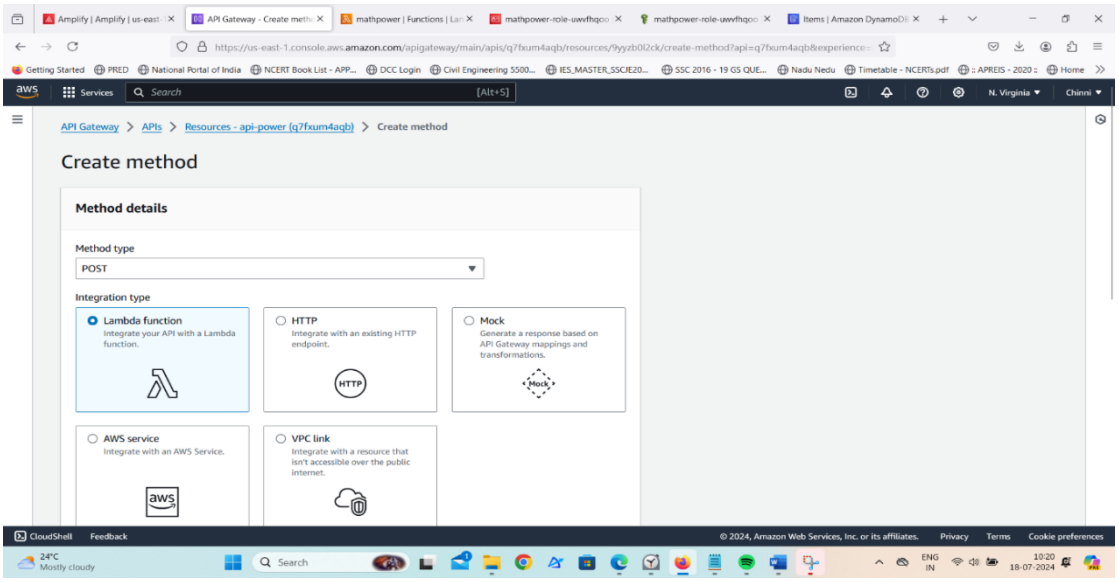


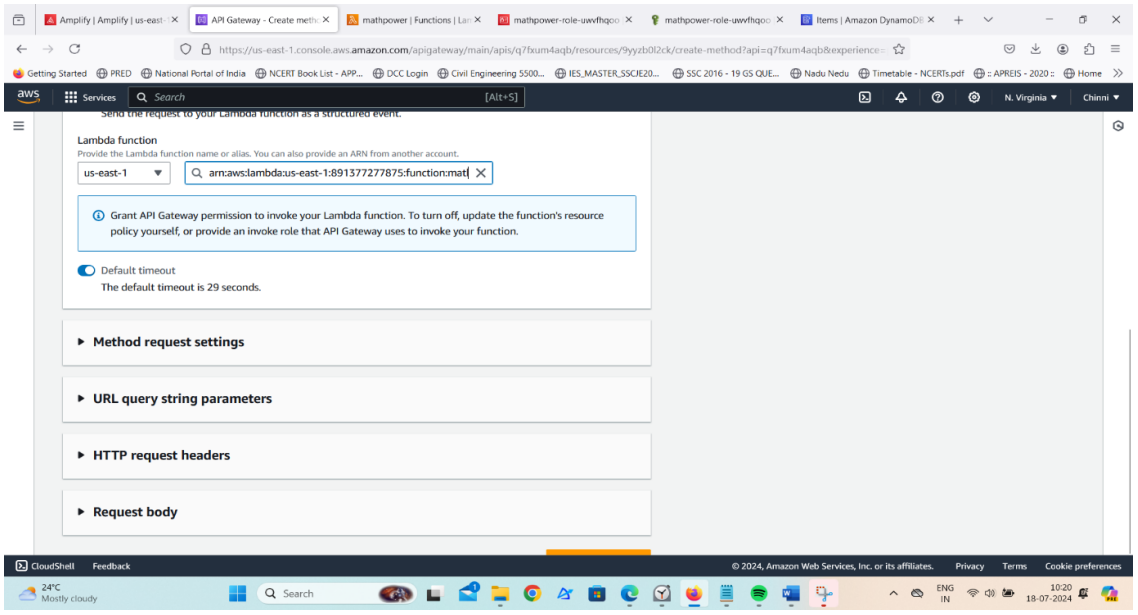
A REST API is an architectural style for networked applications using HTTP requests to access and manipulate resources via URLs.

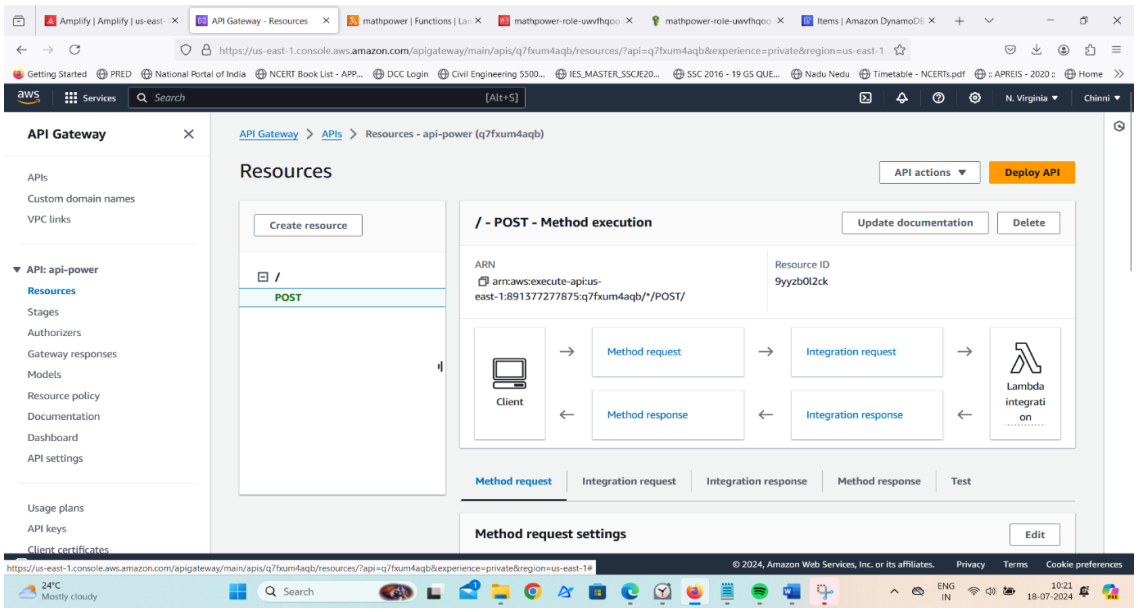
Choose the ‘REST’ protocol for the API, select ‘New API’ under ‘Create new API’ and give the API a name, then click on ‘Create API’

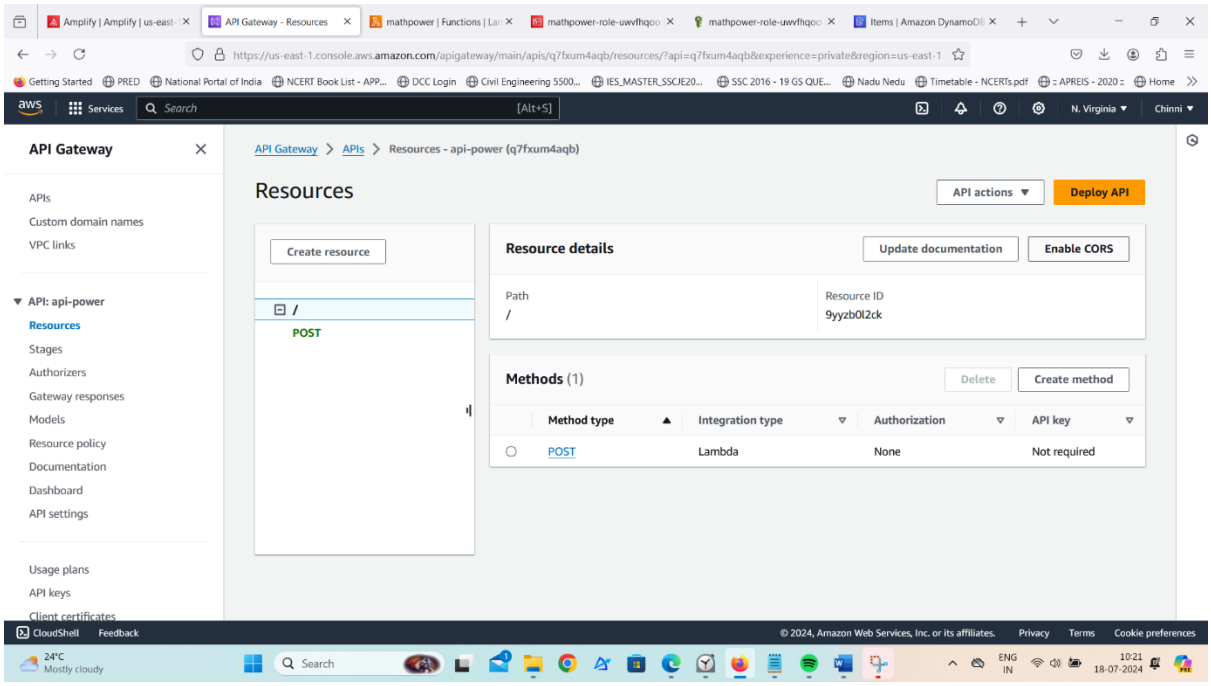


An API POST method is used to send data to a server to create or update a resource.

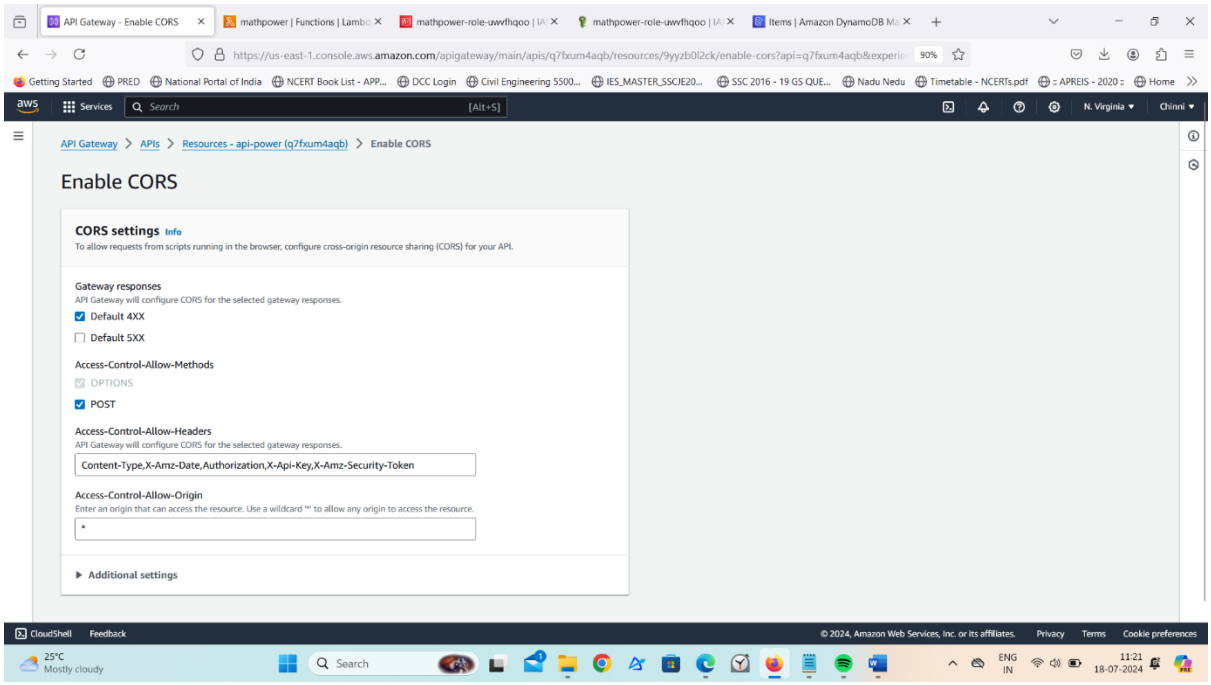

API Gateway > APIs > Resources - api-power (q7fxum4aqb) > Create method
Create method
Method details
Method type
POST
Integration type
Lambda function
Integrate your API with a Lambda function.
HTTP
Integrate with an existing HTTP endpoint.
Mock
Generate a response based on API Gateway mappings and transformations.
AWS service
Integrate with an AWS Service.
VPC link
Integrate with a resource that isn't accessible over the public internet.

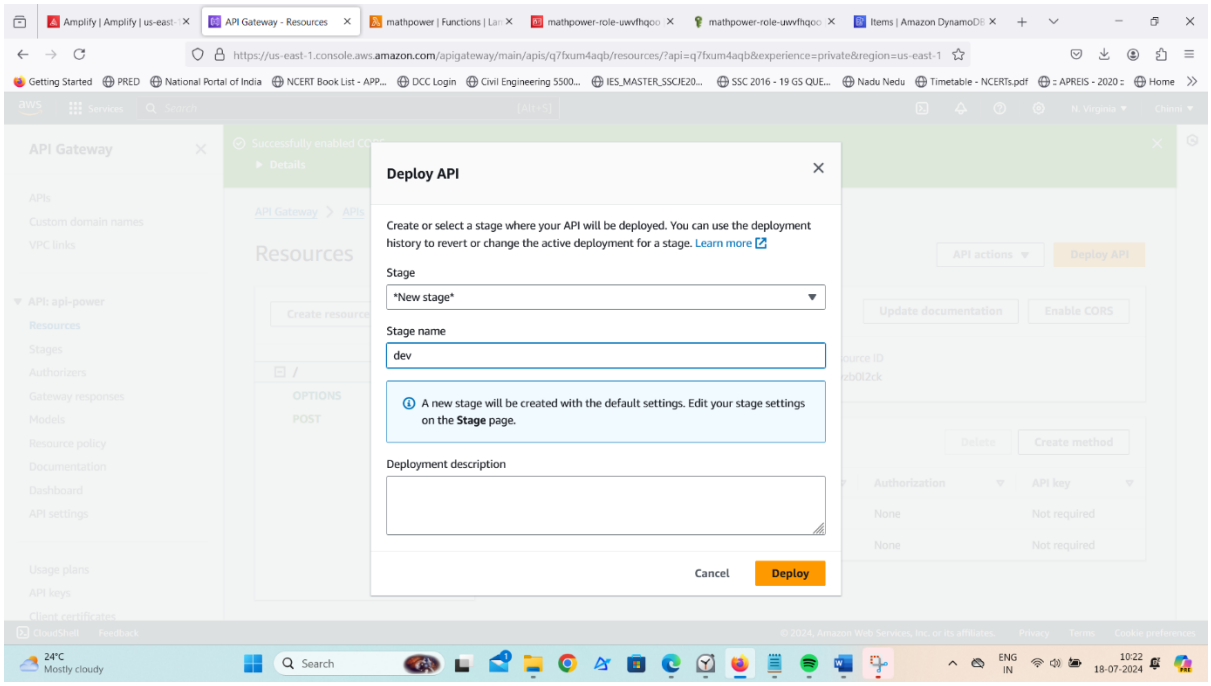

Lambda function
Provide the Lambda function name or alias. You can also provide an ARN from another account.
us-east-1
arn:aws:lambda:us-east-1:891377277875:function:mat
Grant API Gateway permission to invoke your Lambda function. To turn off, update the function's resource policy yourself, or provide an invoke role that API Gateway uses to invoke your function.
Default timeout
The default timeout is 29 seconds.
Method request settings
URL query string parameters
HTTP request headers
Request body


API Gateway
APIs
Custom domain names
VPC links
API: api-power
Resources
Stages
Authorizers
Gateway responses
Models
Resource policy
Documentation
Dashboard
API settings
Usage plans
API keys
Client certificates
API Gateway > APIs > Resources - api-power (q7fxum4aqb)
Resources
API actions
Deploy API
Create resource
/
POST
/ - POST - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:us-east-1:891377277875:q7fxum4aqb/*/POST/
Resource ID
9yyzb0l2ck
Client
Method request
Integration request
Lambda integration
Method response
Integration response
Method request
Integration request
Integration response
Method response
Test
Method request settings
Edit

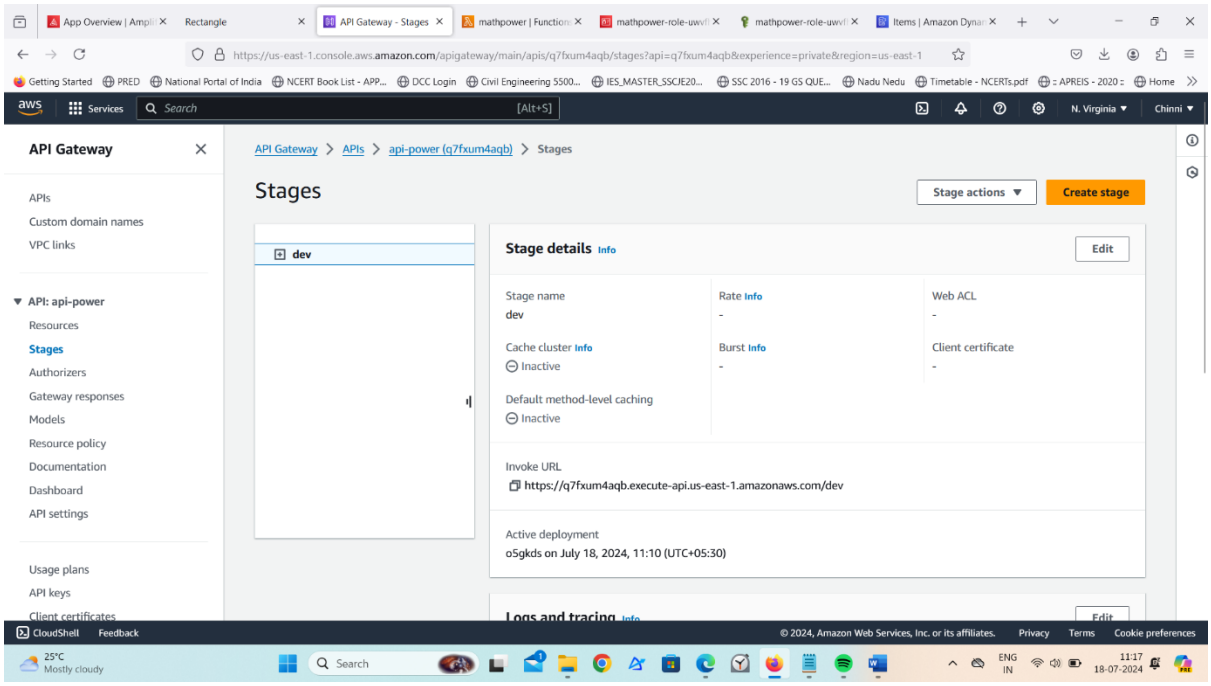


CORS (Cross-Origin Resource Sharing) in API Gateway is a mechanism that allows resources on a web page to be requested from another domain outside the domain from which the first resource was served, enhancing the security of web applications by controlling access to resources.





A stage in API Gateway represents a deployment environment (e.g., development, production) where API configurations and resources can be managed, tested, and accessed.



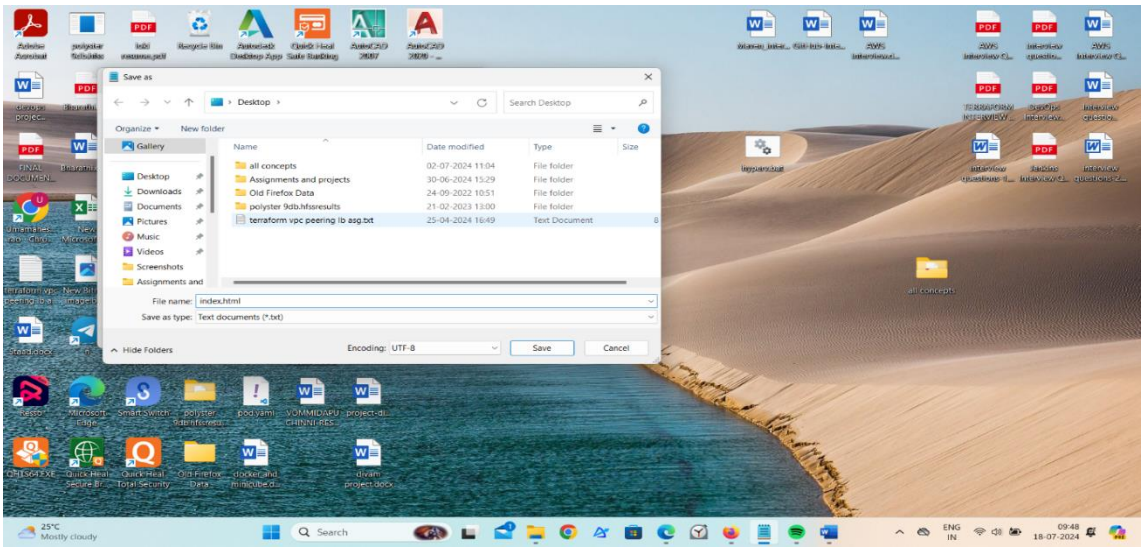
Step4: Setting up frontend web application

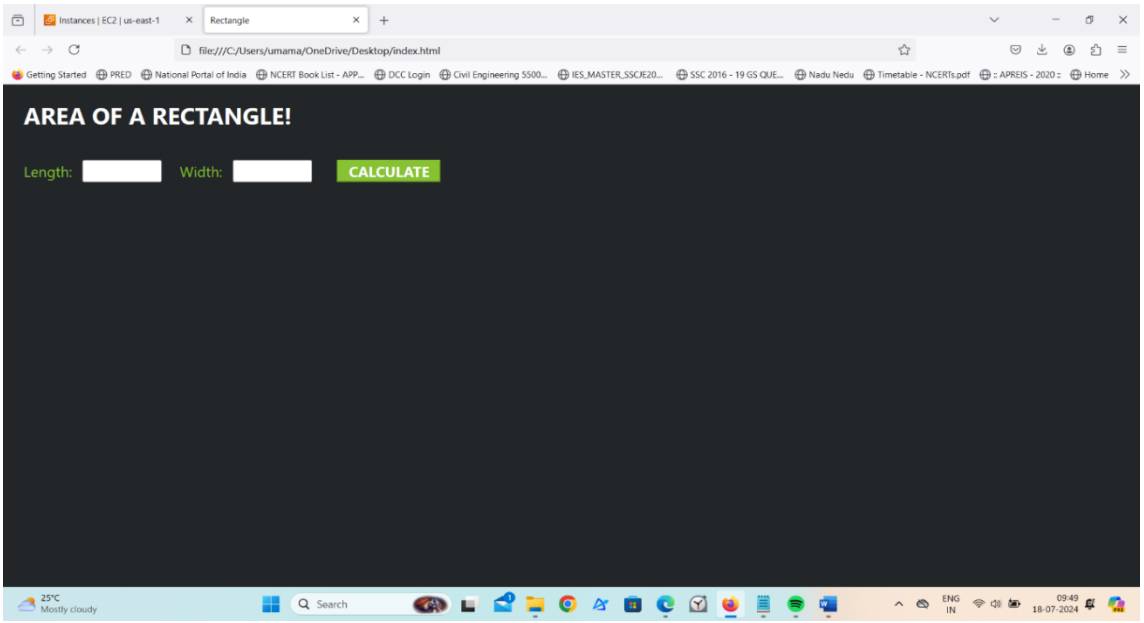
Create a new file in desktop and keep the below code in it.

```
<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>Rectangle</title>

    <!-- Styling for the client UI -->

    <style>

        h1 { color: #FFFFFF; font-family: system-ui; margin-left: 20px; }

        body { background-color: #222629; }

        label { color: #86C232; font-family: system-ui; font-size: 20px; margin-left: 20px; margin-top: 20px; }

        button { background-color: #86C232; border-color: #86C232; color: #FFFFFF; font-family: system-ui; font-size: 20px; font-weight: bold; margin-left: 30px; margin-top: 20px; width: 140px; }

        input { color: #222629; font-family: system-ui; font-size: 20px; margin-left: 10px; margin-top: 20px; width: 100px; }

    </style>

    <script>

        var callAPI = (length, width) => {

            var myHeaders = new Headers();

            myHeaders.append("Content-Type", "application/json");



            // Ensure the length and width are integers
```

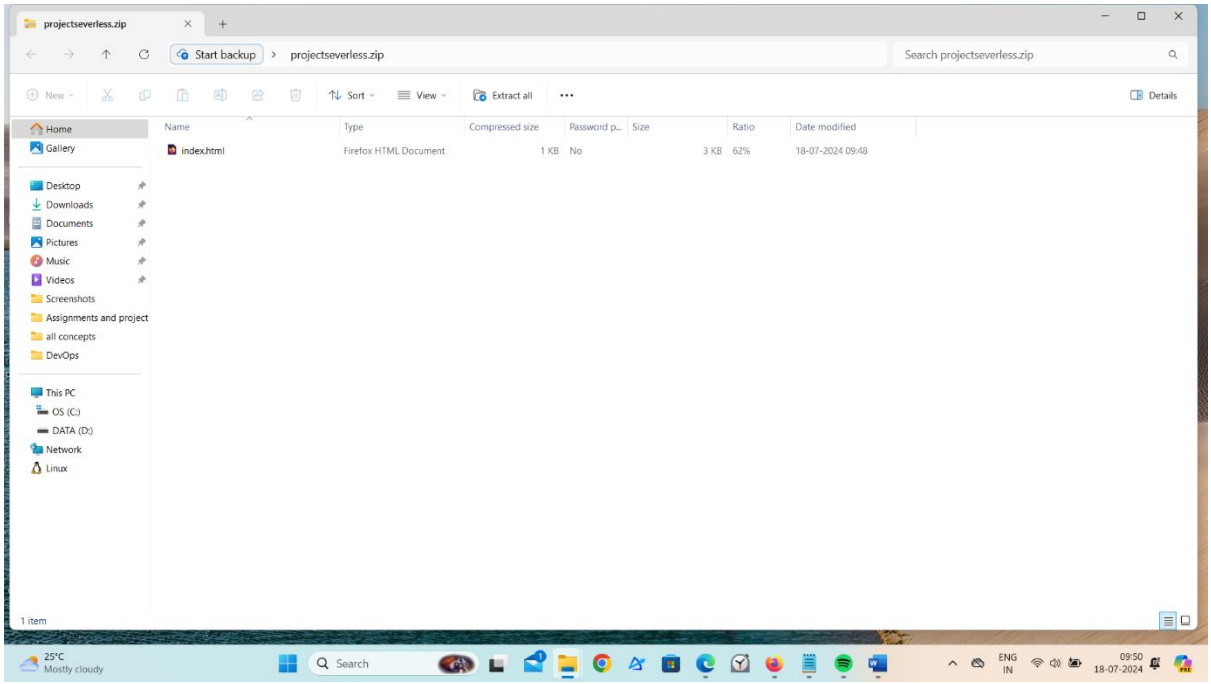
```
        length = parseInt(length);
        width = parseInt(width);

        if (isNaN(length) || isNaN(width)) {
            alert('Length and width must be valid numbers');
            return;
        }

        var raw = JSON.stringify({ "length": length, "width": width });
        var requestOptions = { method: 'POST', headers: myHeaders, body: raw, redirect: 'follow' };

        fetch("https://q7fxum4aqb.execute-api.us-east-1.amazonaws.com/dev", requestOptions)
            .then(response => response.text())
            .then(result => alert(JSON.parse(result).body))
            .catch(error => console.log('error', error));
    }
  </script>
</head>
<body>
  <h1>AREA OF A RECTANGLE!</h1>
  <form>
    <label>Length:</label>
```

```
        <input type="text" id="length">

        <label>Width:</label>

        <input type="text" id="width">

        <button type="button"
onclick="callAPI(document.getElementById('length').value,
document.getElementById('width').value)">CALCULATE</button>

    </form>

</body>

</html>
```



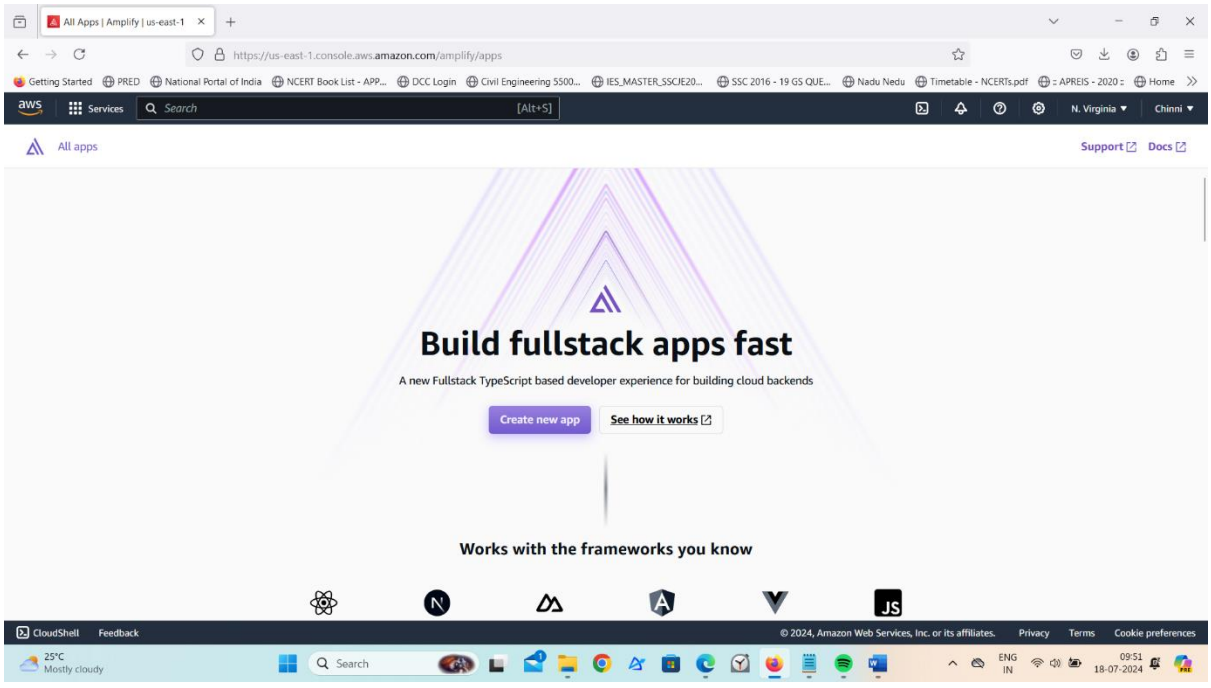


Create a new zip folder on desktop and move that index.html file into that zip folder.

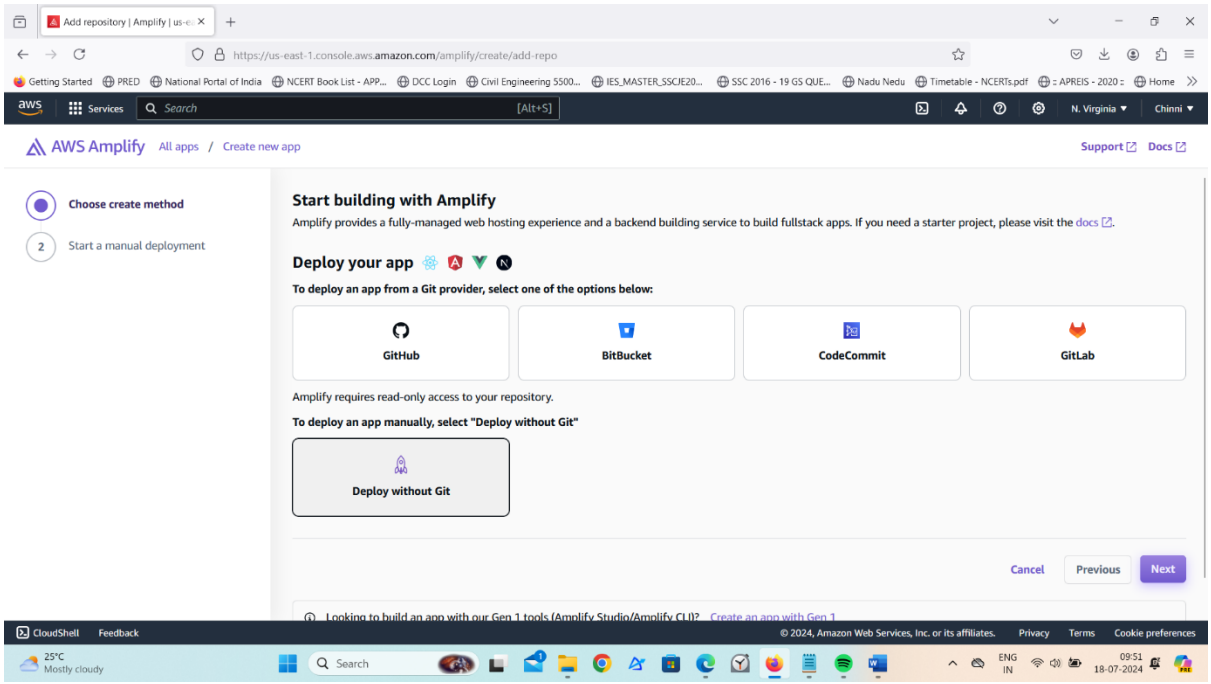


AWS Amplify is a set of tools and services provided by Amazon Web Services (AWS) that enables developers to build scalable full-stack applications quickly. It includes features for frontend web and mobile development, backend services like authentication, APIs, databases, storage, and hosting, all managed through a single unified platform.

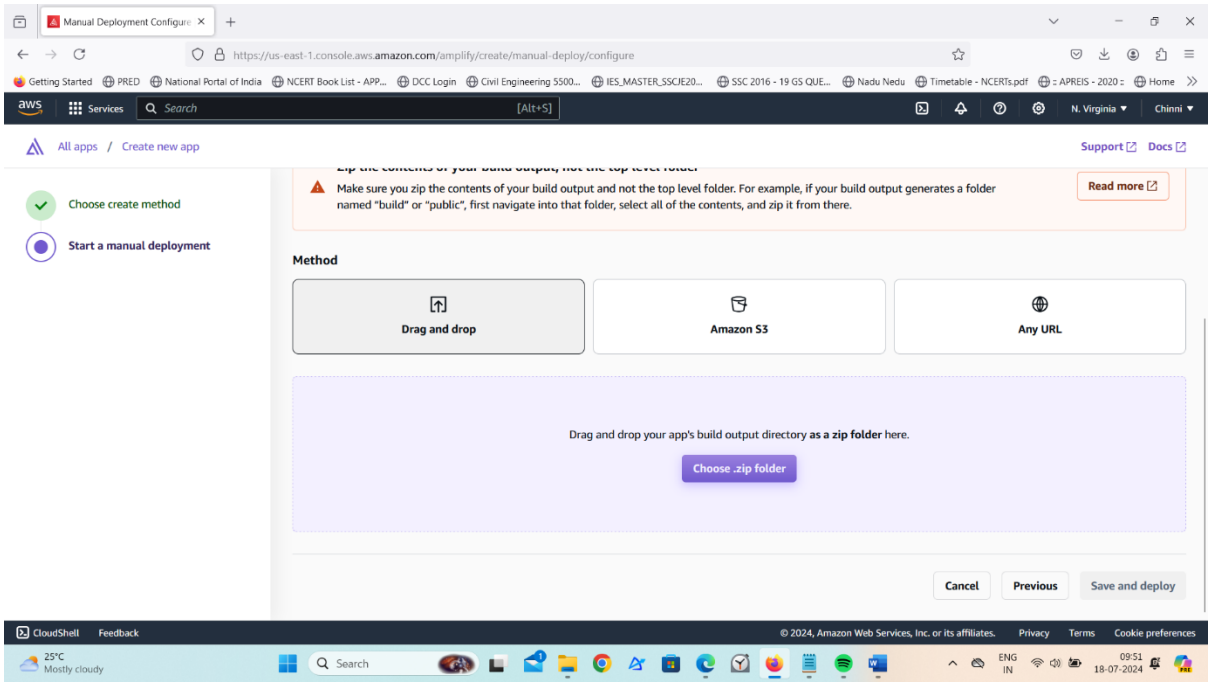
Search AWS Amplify in AWS search tab. Open it, click on create new app.



Select deploy without git, because we kept our code in our own zip folder. Click on next.

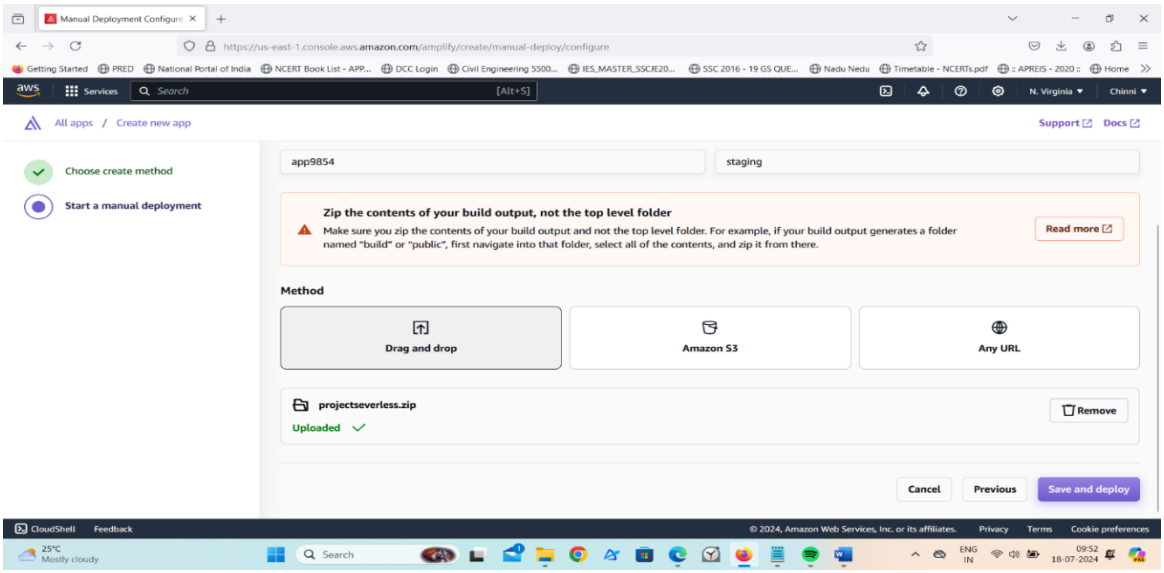


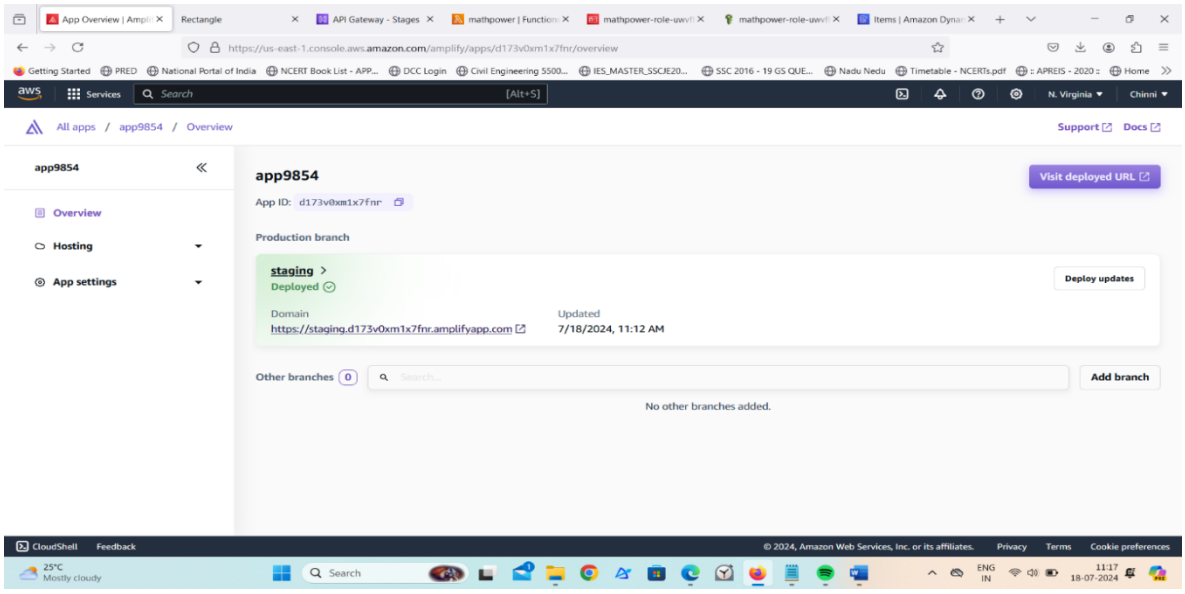
Click on choose.zip folder. Choose our zip folder. Click on open.



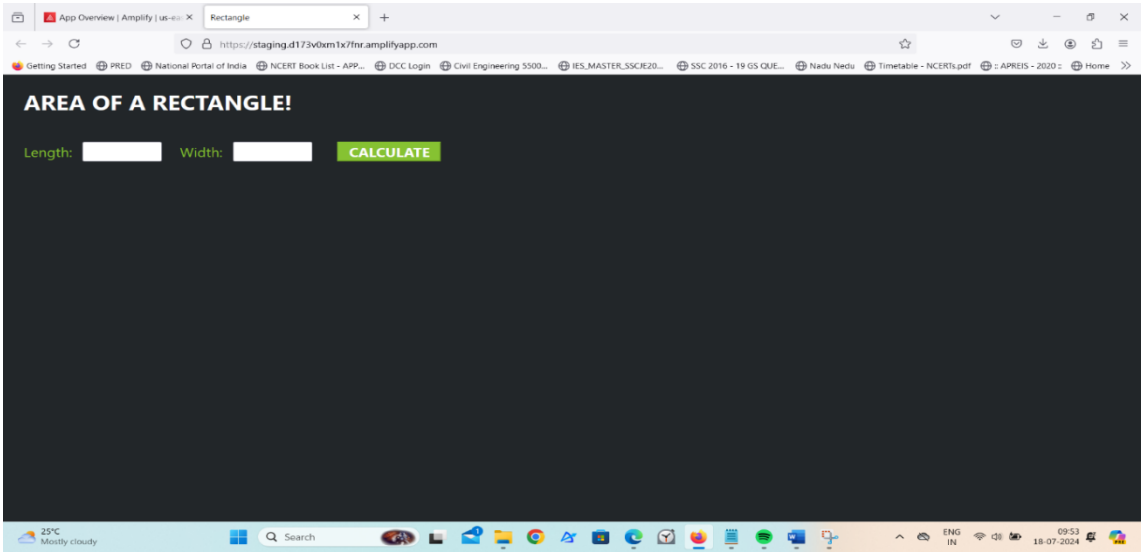
Click on save and deploy.

Wait until its available.

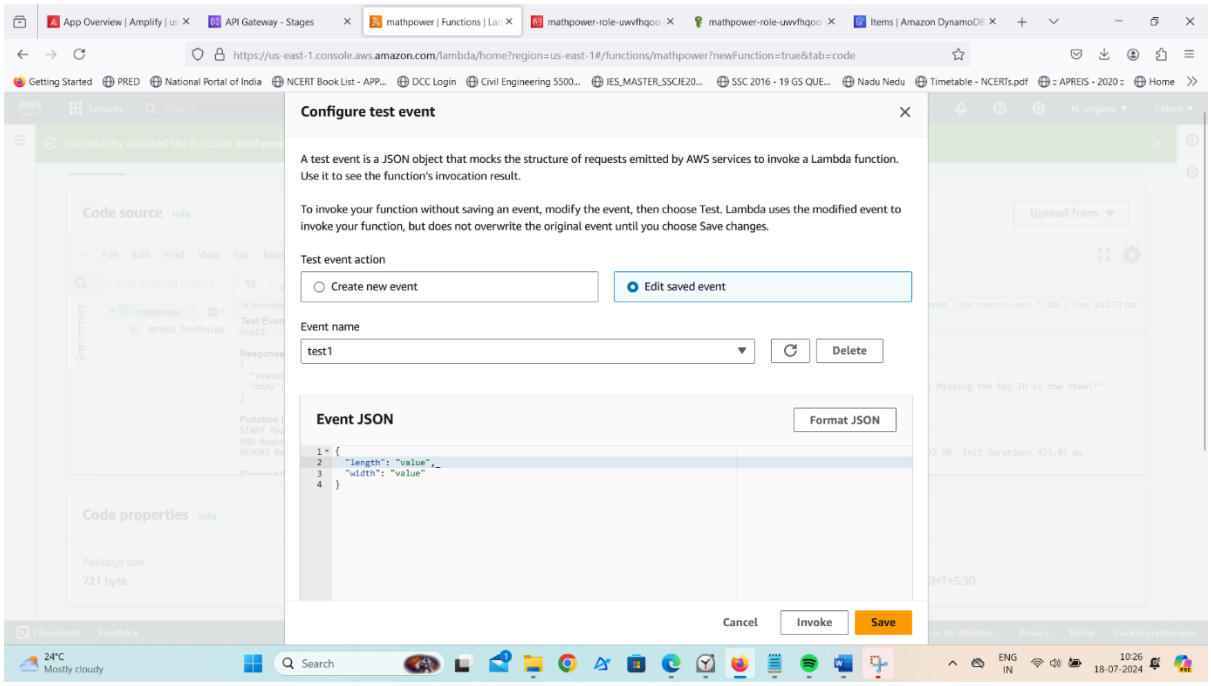




Click on domain we see our frontend web app.



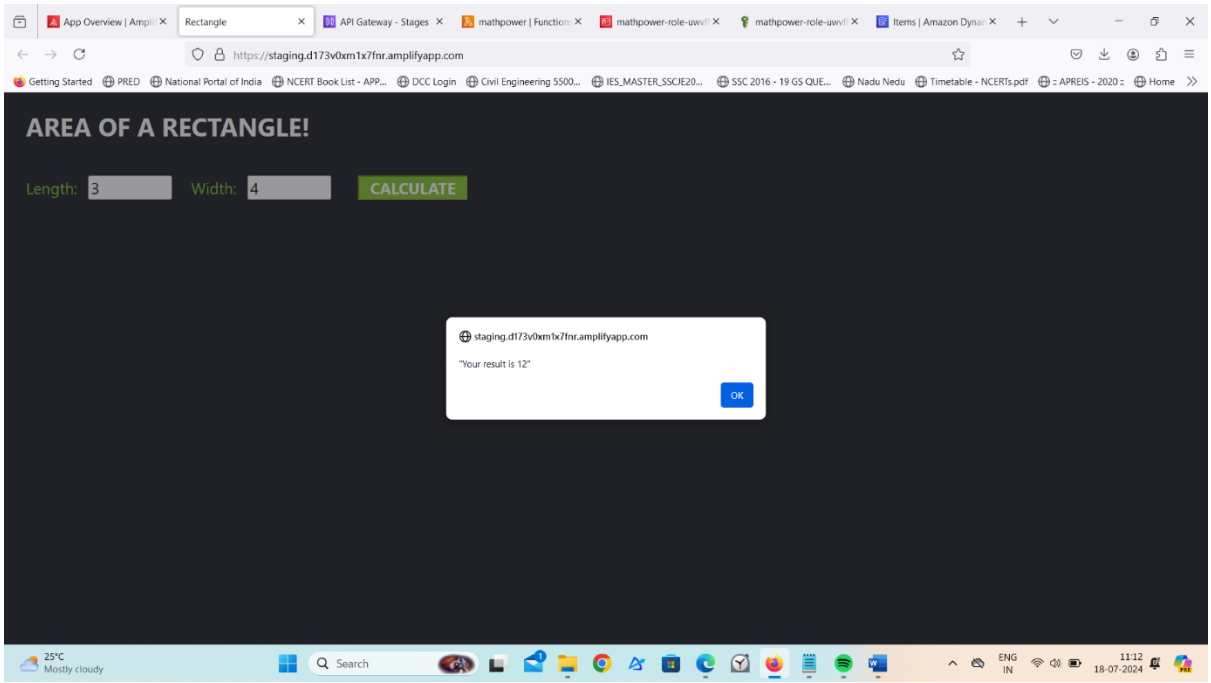
Now go to lambda click on test side button we see configure event test click on it and change to values in EVENT JSON. Click on save.

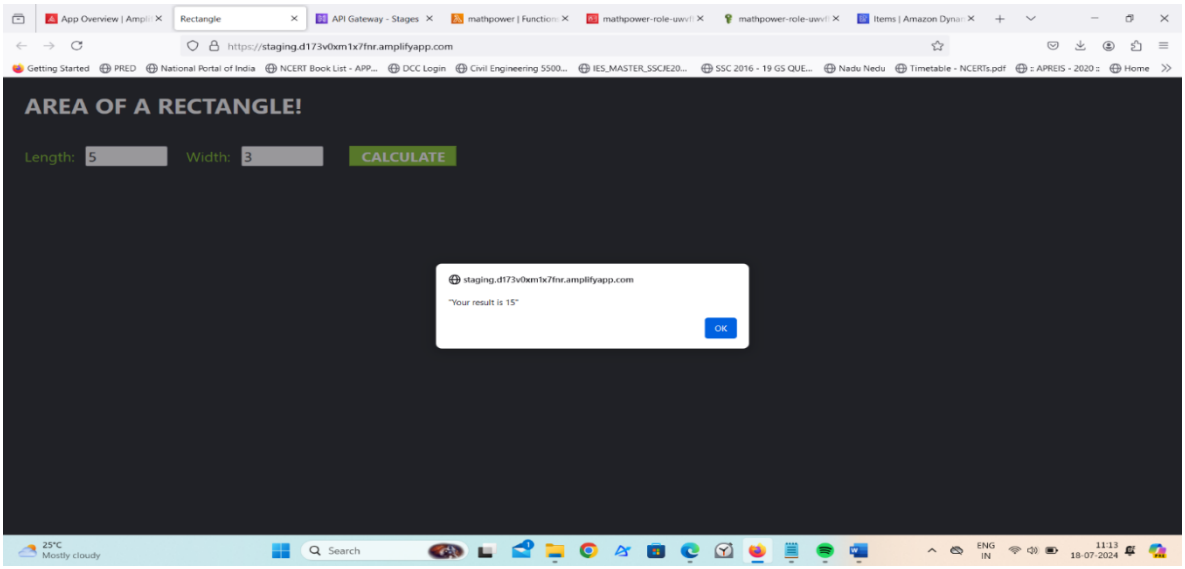


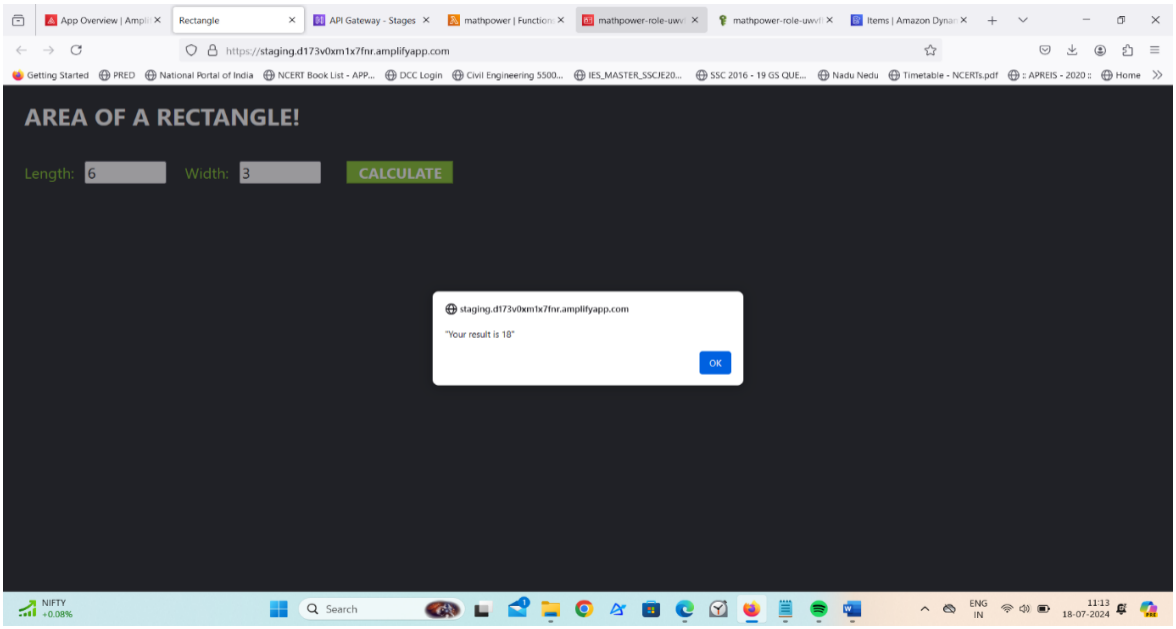
Now go to amplify and click on domain.

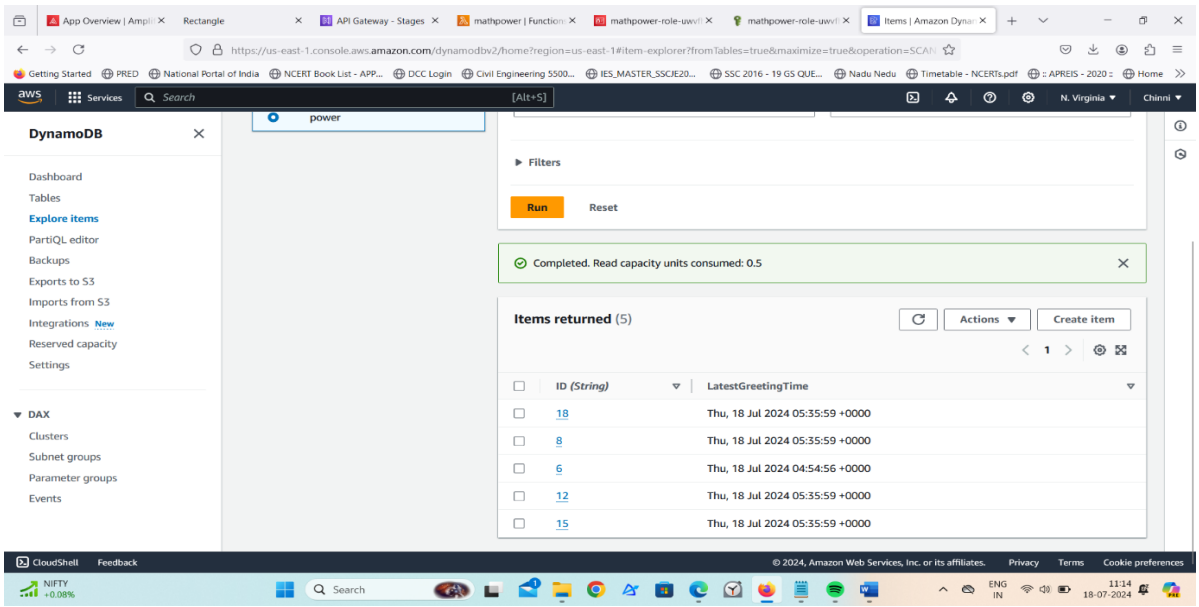
Give input values in the length and width space. click on calculate, It will calculate the area of rectangle.

The result is shown at their only.

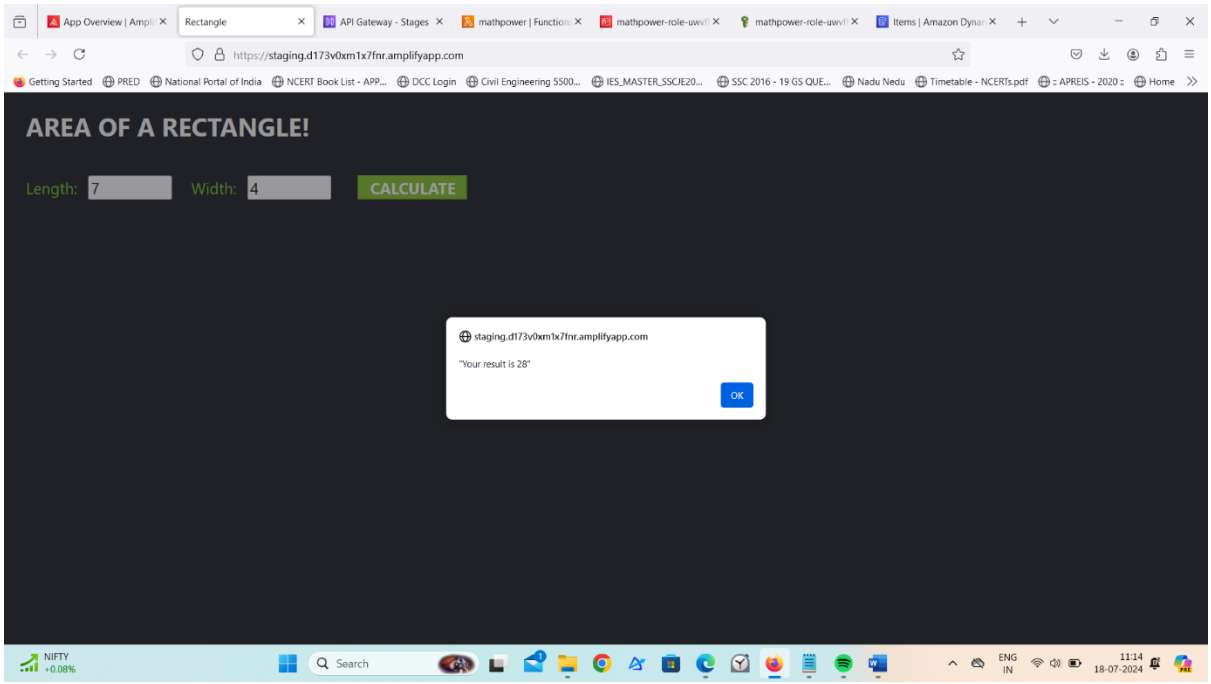


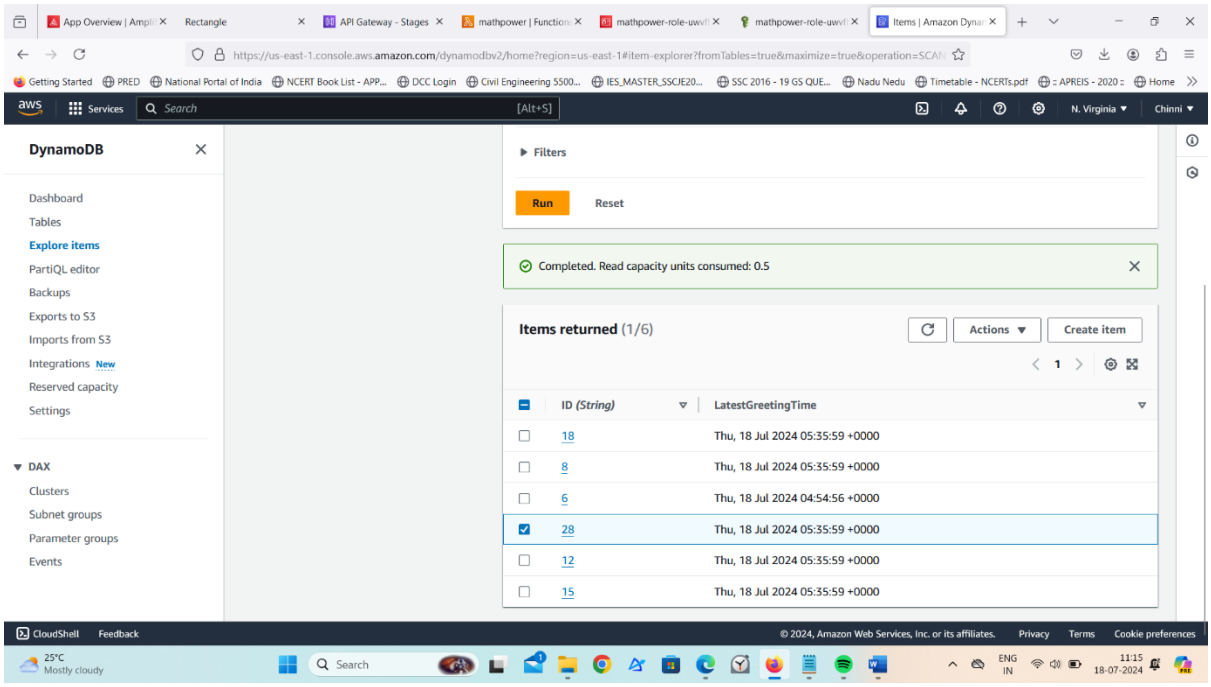

AREA OF A RECTANGLE!
Length: 5
Width: 3
CALCULATE
staging.d173v0xm1x7fnr.amplifyapp.com
"Your result is 15"
OK

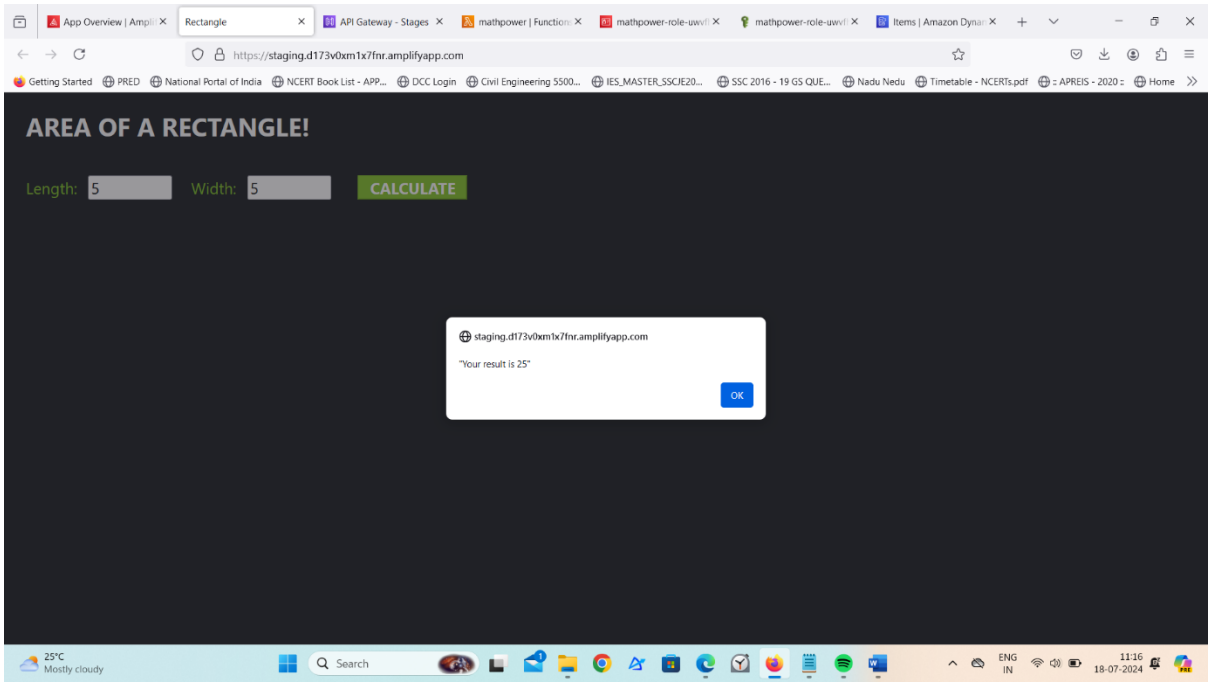

AREA OF A RECTANGLE!
Length: 6
Width: 3
CALCULATE
staging.d173v0xm1x7fnr.amplifyapp.com
"Your result is 18"
OK

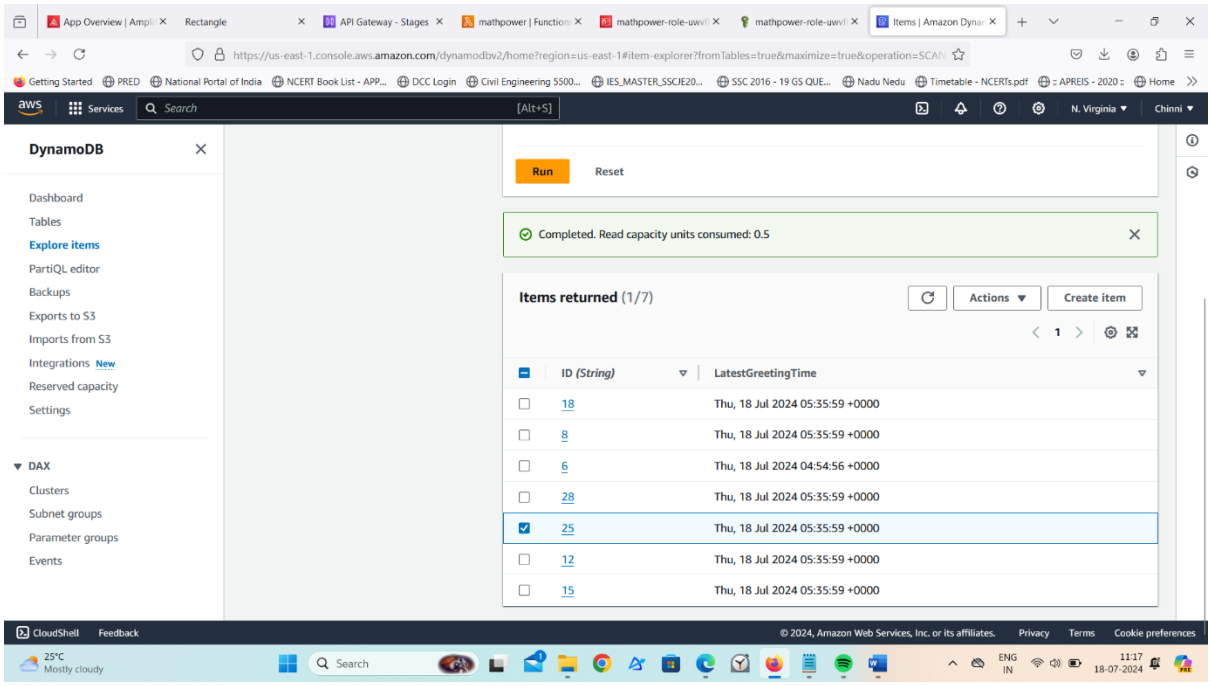

DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings
DAX
Clusters
Subnet groups
Parameter groups
Events
Filters
Run
Reset
Completed. Read capacity units consumed: 0.5
Items returned (5)
Actions
Create item
ID (String)
LatestGreetingTime
18 Thu, 18 Jul 2024 05:35:59 +0000
8 Thu, 18 Jul 2024 05:35:59 +0000
6 Thu, 18 Jul 2024 04:54:56 +0000
12 Thu, 18 Jul 2024 05:35:59 +0000
15 Thu, 18 Jul 2024 05:35:59 +0000

We see that our results are stored in our dynamo DB table.









Destroy/delete all created services

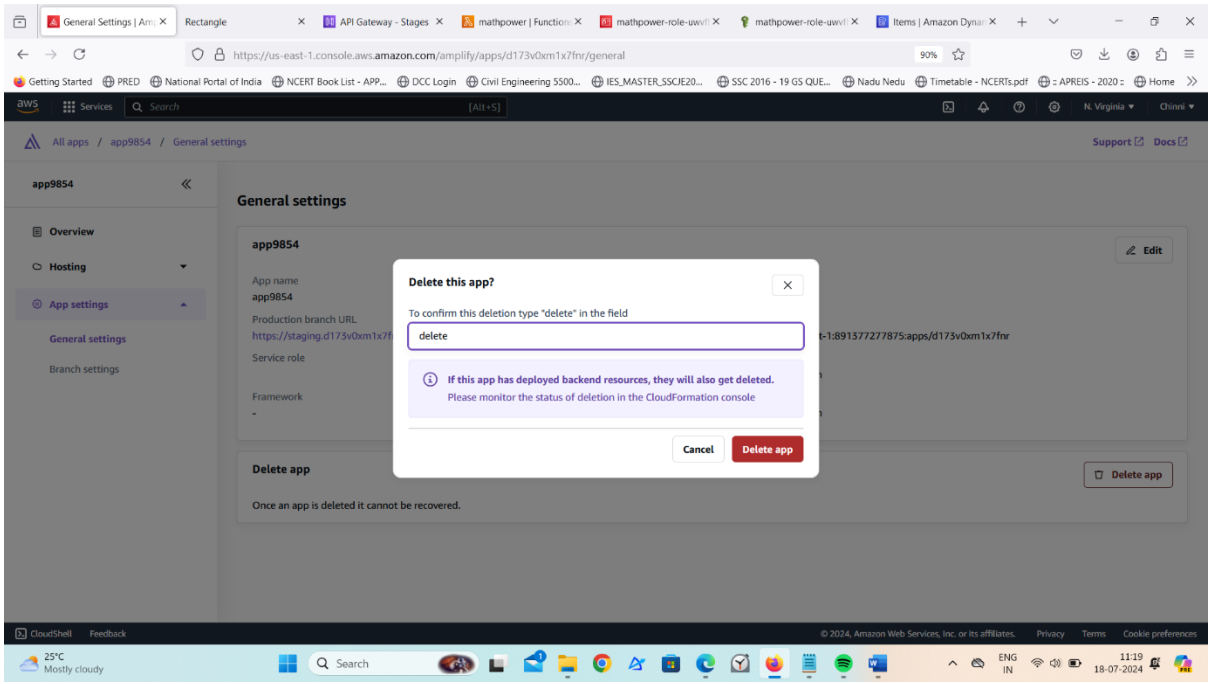
Remember to delete your resources to avoid unnecessary charges:

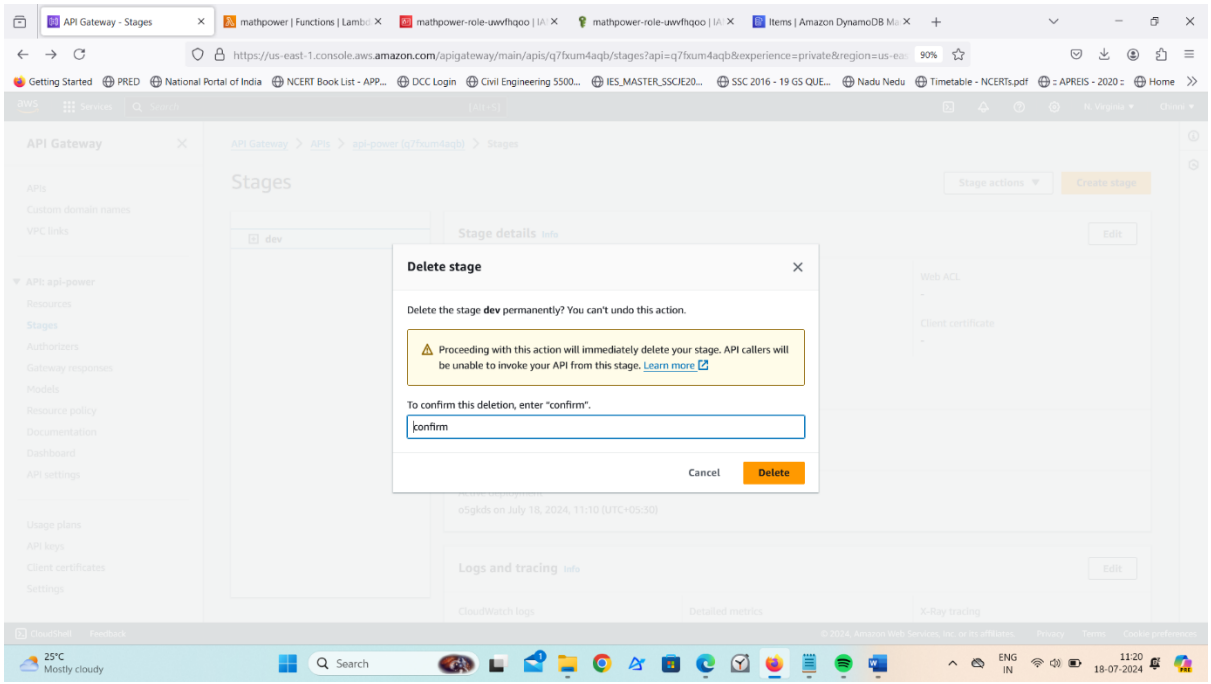
Delete the Amplify App

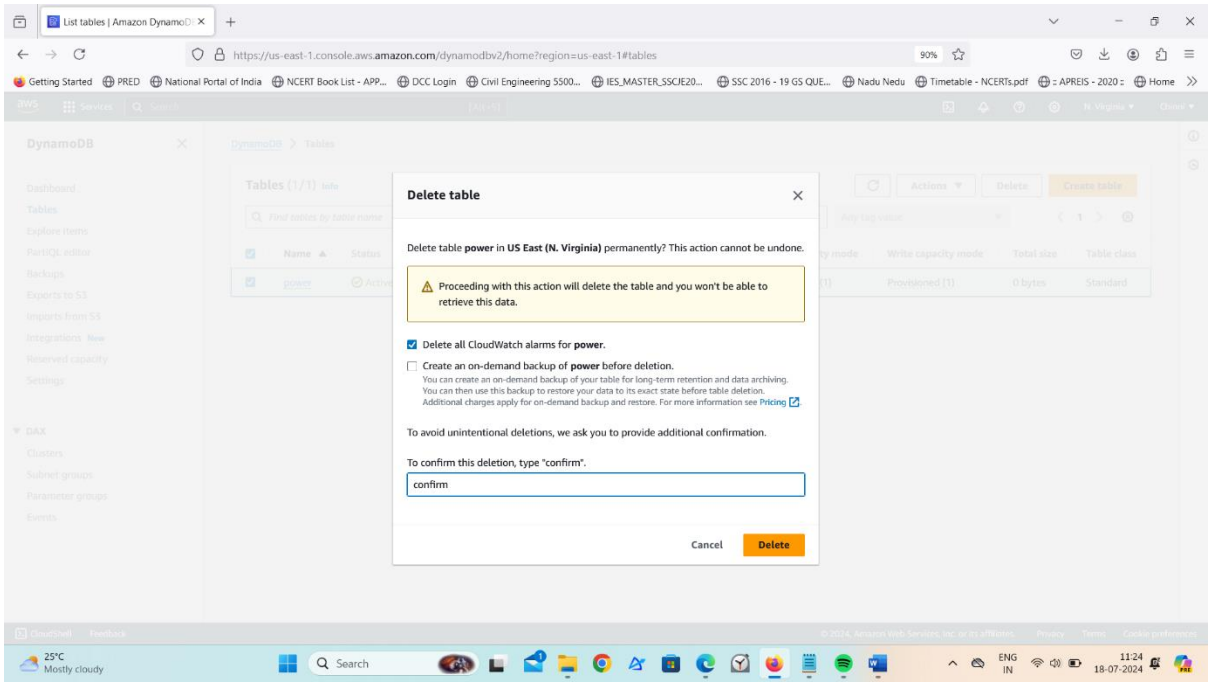
Delete the DynamoDB Table

Delete the Lambda function

Delete the API Gateway







Conclusion:

In this guide, we've shown how AWS Amplify, DynamoDB, Lambda, and API Gateway can make serverless web app development easy and efficient. By using these tools, you can build powerful applications without worrying about managing servers.

AWS Amplify helps you quickly set up authentication, storage, and APIs. DynamoDB provides a fast and scalable database. Lambda lets you run your code in the cloud without managing servers, and API Gateway makes it simple to create and manage APIs.

Using these services together, you can create web applications that are scalable, cost-effective, and easy to maintain. With the knowledge from this guide, you're ready to start building serverless applications that can grow with your needs. Enjoy the simplicity and power of serverless development with AWS!